Virginia
CENTER of
EXCELLENCE
for Software Reuse and Technology Transfer

October 16, 1992

Mr. Dan Fesko
Administrative Grants Office (AGO)
DARPA/DASB Library
Office of Naval Research Resident Representative
2135 Wisconsin Ave. NW, Suite 102
Washington DC 20007-2259

Subject: Special Technical Report No. 3
Re: Grant MDA972-92J-1018

Dear Sir:

In accordance with the referenced grant the Virginia Center of Excellence hereby submits the following special report documentation:

1. Synthesis Guidebook: Volume 1-Methodology Definition; Volume 2- Case Studies

- Reference SPC-92111-CMC, Version 01.00.00

This is the third delivery made for the subject grant under Section 9.1.b. If you have any questions, please contact me at (703) 742-7172 or Mr. Ed Evers at (703) 742-7113.

Sincerely,

Edward B. Louy
Manager, Accounting

/djk
Enclosures
cc: Project Office
Dr. Jack Kramer
DARPA/SISTO
801 North Randolph Street
Suite 400
Arlington VA 22203

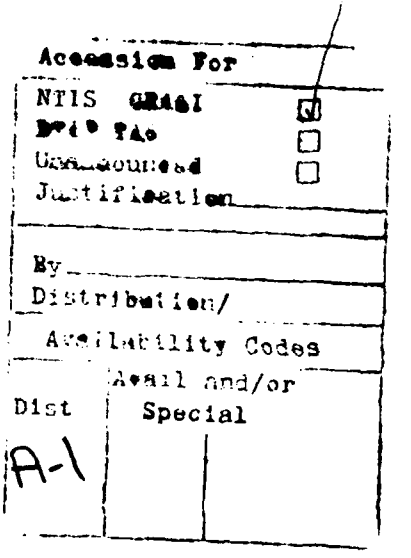
SPC BUILDING 2214 ROCK HILL ROAD HERNDON, VA 22070 (703) 742-7300 FAX (703) 742-7200

Mr. Dan Fesko
Administrative Grants Office (AGO)
October 16, 1992

Defense Technical Information Center
Attn: DTC-FDAC
Cameron Station
Alexandria VA 22304-6145

ST-A per telecon, Dr. Kramer,
DARPA/SISTO. Arl., VA 22203

12-15-92 JK



SYNTHESIS GUIDEBOOK
VOLUME 1
METHODOLOGY DEFINITION

SPC-92111-CMC

VERSION 01.00.00

OCTOBER 1992

SYNTHESIS GUIDEBOOK
VOLUME 1
METHODOLOGY DEFINITION

SPC-92111-CMC

VERSION 01.00.00

OCTOBER 1992

SOFTWARE PRODUCTIVITY CONSORTIUM
SPC Building
2214 Rock Hill Road
Herndon, Virginia 22070

CONTENTS

FIGURES

TABLES

This page intentionally left blank.

ACKNOWLEDGEMENTS

Grady Campbell, Jim O'Connor, and Neil Burkhard were the primary authors of this guidebook. Jeff Facemire and Rich McCabe wrote, reviewed, or otherwise made major contributions to particular sections.

The Consortium wishes to thank the participants in the Rockwell pilot, particularly Catharine Mansour, for helping to refine the ideas of Synthesis and for their many suggestions for improving the guidebook.

Roger Williams, Stuart Faulk, and Wil Spencer served as primary reviewers and helped to improve many aspects of the guidebook. Mike Cochran, Charlotte Winnick of United Technologies, Tim Longenecker of Boeing, Steve Wartik, and Dave Weiss also provided valuable comments.

Leslie Hubbard suggested many improvements in the structure and form of the document. The Environment and Support Services group of the Software Productivity Consortium provided superb help at all stages in producing the document.

This page intentionally left blank.

PART I: OVERVIEW

This page intentionally left blank.

OV.1. INTRODUCTION

Synthesis is a *methodology* for constructing software *systems* as instances of a *family* of systems that have similar descriptions (Campbell, Faulk, and Weiss 1990). This guidebook provides an introduction to the practice of the Synthesis methodology of software development. As you understand the essential similarities and variations in the systems you build, Synthesis will enable you to exploit those similarities to eliminate redundant work and thus to focus your effort on resolving variations to satisfy the needs of your organization and its customers.

Synthesis focuses on your need both to deliver high quality *products* to customers and to accomplish this profitably. To this end, Synthesis defines two *processes*: *Application Engineering* and *Domain Engineering*. Application Engineering is how a group (or *project*) in your organization creates a product to meet *customer requirements*. Domain Engineering is how your organization improves productivity by creating an Application Engineering process, tailored for projects in your business area and supporting standardized, reusable products. Since projects in the same business area tend to build systems that satisfy similar needs, these systems can be thought of as instances of a family. A family of systems is a basis for a flexible approach to standardization that can accommodate diverse and changing needs. A business area whose objectives are fulfilled by a family is referred to as a *domain*.

Both the mission of organization and the changing needs of its customers determine the objectives of that *business-area* (i.e., product-line) *organization*. Synthesis is a comprehensive, business-area-level solution to problems of software productivity and quality in the building of systems to meet diverse and changing needs. Synthesis is a systematic approach to software development founded on the belief that the resources of a business-area organization should be managed not only to meet the immediate needs of customers, but also as an investment in future capability.

1.1 DOCUMENT PURPOSE, SCOPE, AND AUDIENCE

This guidebook describes a sound approach for effective family-oriented software development. It serves as a detailed guide to the practice of Synthesis and helps you begin to use this approach. As you gain experience in using Synthesis you will be able to refine and modify this guidance to meet the specific needs of your company and business area more effectively.

The scope of this guidebook includes all activities and products related to the production of software and support of the needs and objectives of a business-area organization and its customers. The starting point of Synthesis activity is the establishment of organizational *business objectives*. In addition, various Synthesis activities require you to standardize management, requirements, design, implementation, and verification and validation practices throughout your organization. Synthesis is an integrating framework for the *methods* you choose to standardize your practices. While detailed descriptions of some of these methods are outside the scope of this guidebook, they are available, often in other Consortium publications of the Software Productivity Consortium (referenced in relevant

sections of the guidebook. Standardization of these activities in terms of both form and content, using methods of your choice is essential to an effective Synthesis practice.

The audience for this guidebook includes business-area managers, project managers, and engineers of all disciplines who work together to accomplish the objectives of a business-area organization. The audience is assumed to be well grounded and experienced in the standard (or prevailing) software development methods used in their organization but to have no experience with Synthesis. Practical use of this release of the guidebook requires the participation of technologists who have completed a three-day, Consortium-sponsored training class. With Consortium assistance, these technologists advise participating managers and engineers by interpreting and elaborating on the guidebook to meet specific needs. The conduct of Consortium-assisted pilot projects is the requisite first step in transitioning to production use of Synthesis.

1.2 DOCUMENT STATUS AND EVOLUTION

This first release of the guidebook ends the initial year of a projected four-year effort to produce a guidebook to be used by business-area organizations (or refined by technologists to meet particular needs). Appendix A describes the levels of maturity through which the guidebook and each of its sections will progress, and the current status of each. (Note that some sections are omitted from this release.) Member-company pilot projects are the Consortium's primary means for refining the content and quality of the guidebook.

The *Synthesis Methodology Reference Model* (Campbell 1990a) is a canonical definition of the processes, products, and *activities* of Synthesis. The Reference Model defines the essentials of Synthesis for methodologists. This guidebook both elaborates on that definition and provides a practical guide for the use of Synthesis. Because of insights gained over the last year in explaining and using Synthesis, some of the concepts and terminology used in this guidebook differ from those in the original Reference Model document. Appendix B defines the relationship between Reference Model and guidebook concepts and terminology.

1.3 RELATED PUBLICATIONS

Publication of this guidebook follows two years of work on Synthesis at the Consortium. In addition to the *Synthesis Methodology Reference Model* noted above, the Consortium has published a continuing series of products that describe different aspects of Synthesis. These include:

- *Systematic Reuse—The Competitive Edge* (Software Productivity Consortium 1991d), a video that briefly explains the concepts and rationale for Synthesis with an orientation to the concerns of executive management.
- *Introduction to Synthesis* (Campbell, Faulk, and Weiss 1990), an informal explanation of the Synthesis vision and its foundations. This report should be read before the guidebook.
- *Synthesis Program—A Risk Analysis* (Campbell 1990b), which identifies the *risks* that must be mitigated before Synthesis is accepted for routine production use in member companies.
- *Synthesis Operational Scenarios* (Weiss 1990), which explores various scenarios in which Synthesis might be used.

- *Synthesis Transition Strategies* (Williams 1990b), which discusses a strategy for the incremental transitioning of Synthesis into production use, taking into consideration the particular needs of a member company and its customers.

- *Member Company Domain Characterization for Synthesis* (Williams 1990a), which discusses the issues of identifying viable *application* domains for member companies.

- *Synthesis Economic Analysis and Reuse Economic Model Improvements* (Cruickshank and Gaffney 1990), which proposes a preliminary economic model for Synthesis derived from experience in parts-oriented reuse. *The Economics of Software Reuse* (Cruickshank and Gaffney 1991) refines and extends this model for systematic reuse in general.

- *Applying Synthesis in the Design Composer Domain* (Burkhard et al. 1990), which illustrates the application of a pre-guidebook understanding of Synthesis to a family of design composer tools.

- *Synthesis Status Report: Fourth Quarter 1990* (Wartik et al. 1990), which, in part, presents the results of an extended investigation of alternative technologies for the representation of *component* families.

- *An Application Generator Tool Evaluation Method* (Jenkins and Facemire 1991), which discusses the use of Synthesis techniques to derive requirements for an ideal application generator tool to meet particular needs and, together with Analytic Hierarchic Process techniques, to evaluate and refine commercial tools to approximate that ideal.

- *Synthesis Seminar Notebook* (Software Productivity Consortium 1991e), used in a three-day training class that introduces the concepts and practice of Synthesis to a novice audience.

- *TRF2 Metaprogramming Tool User Guide* (Software Productivity Consortium 1991b), which describes a *metaprogramming* notation and tool for creation and use of *abstract components*.

Much of the information about Synthesis is evolving as Consortium and member company experience accumulates. Although earlier products do not reflect recent experience, they still provide many insights into the use of Synthesis. As time permits, the Consortium plans to revise many of these products to reflect continuing experience.

1.4 DOCUMENT STRUCTURE

The guidebook is organized into two volumes. Volume 1 includes four parts:

- Overview (OV)

- Domain Engineering (DE)

- Application Engineering (AE)

- Advanced Topics (AT)

Volume 2 includes Case Studies (CS).

Volume 1 includes Part I, Overview; Part II, Domain Engineering; Part III, Application Engineering and supporting glossary, references, bibliography, and index. Part I, Overview consists of this introduction and a description of the context and principles of Synthesis practice. The remaining sections of this introduction define the structure of the rest of the guidebook and the standard notations and conventions used throughout.

Part II, Domain Engineering, defines the activities you follow to standardize the process, *work products*, and practices of Application Engineering for a business-area organization. The activities of Domain Engineering are organized and presented hierarchically (as depicted on page OV.1-5). A description of an aggregate activity is primarily a summarization and roadmap for its subactivities. Each aggregate activity and its subactivities are described in its own section. Each activity is described according to the outline shown in Table OV.1-1.

Part III, Application Engineering, defines a prototypical process, activities, and products for Application Engineering. A primary objective of Domain Engineering is to refine this definition to satisfy the needs and objectives of supported projects. Each activity of Application Engineering is also described according to the outline shown in Table OV.1-1.

Part IV, Advanced Topics, is reserved for future use in presenting discussions of issues requiring advanced understanding of Synthesis. The emphasis of such discussions will be on the refinement of Synthesis and its guidebook description to meet particular needs.

1.5 USING THIS GUIDEBOOK

This version of the guidebook is intended primarily as a reference for the use of technologists in supporting Synthesis pilot projects. As a rule, detailed sections are meant to be sufficiently complete for full-scale use. For smaller, more exploratory pilot projects, you may decide to expend less effort on activities related to management, process definition, or opportunities for automation; however, no activities should be skipped entirely.

The guidebook is also organized for use as a graduated introduction to Synthesis. A reading of Sections OV.2, DE.1, and AE.1 is sufficient to gain a basic understanding of the concepts of Synthesis. The hierarchy on page OV.1-5 serves as a guide to which sections you should read for a deeper understanding of particular aspects of Domain Engineering. You may choose to skip sections lower in the hierarchy when more detail is not of interest.

Hierarchy of Domain Engineering Activities

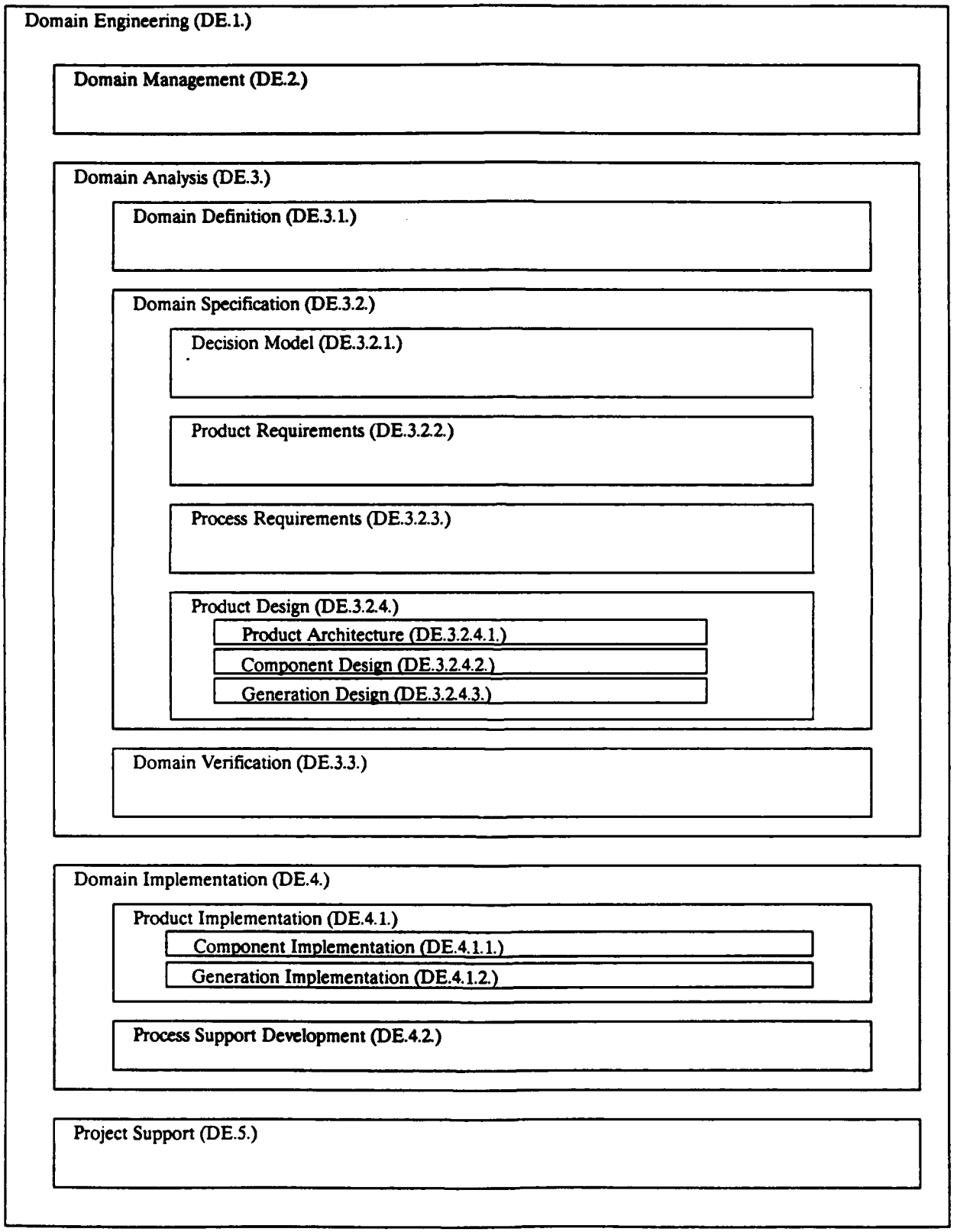


Table OV.1-1. Activity Description Outline

1. ***Getting Started.*** Describe when the activity is relevant and can be performed.
   - Objectives: Explain the objectives that you should achieve in the performance of this activity.
   - Information needs: Identify what information must be available (i.e., baselined) before you can initiate this activity. These are usually products of other Synthesis activities.
   - Experience and expertise needs: Indicate what expertise and experience you need to accomplish the required tasks of this activity most effectively.
2. ***Product Description.*** Describe the work product that results from completion of the activity.
   - Purpose: Describe what is accomplished by producing this work product.
   - Content: Describe the information content of this product.
   - Form and structure: Describe the structure of the product and the form in which its content is to be presented.
   - Verification criteria: Describe how the consistency/completeness, correctness, and quality of the product is to be judged. Provide review questions and metrics that support that evaluation.
3. ***Process Description.*** Describe a process that achieves the objectives of the activity.
   - Procedure: Describe a set of partially-ordered steps that constitute the essentials of the process.
   - Heuristics: For each step of the procedure, suggest techniques that may make performance of the step more effective.
   - Risk management: Identify risks that may arise to prevent successful, timely completion of the activity, and describe strategies for mitigating those risks. Use checkpoints, reviews, and metrics to reveal flaws and misconceptions.
4. ***Interaction With Other Activities.*** Describe interactions that may occur with other activities as a result of attempted use of a work product:
   - Describe *feedback* to activities that provide needed information.
   - Describe feedback from other activities with regard to this activity's products.

   Describe how those interactions stimulate product evolution. For each potential problem, describe:
   - Contingency: The nature of the problem that may be found in the use of a work product.
   - Source: The activity that provides/uses the work product of concern.
   - Response: The appropriate response to the contingency within this activity.
5. ***Advanced Topics.*** (Reserved for future use only.) Provide guidance on complex aspects of this activity in short, topical discussions. More expansive papers will appear in Part IV. Tailoring to particular needs and the use of alternative, immature, but potentially more robust approaches may be discussed here.

1.6 CONVENTIONS AND NOTATION

This guidebook uses the following typographic conventions:

Serif font	General presentation of information.
Capitalized Serif font	Names of Synthesis work products and activities.
CAPITALIZED SERIF font	Names of undesired events.
Italicized serif font	Words, expressions, abbreviations, and acronyms found in the Glossary, mathematical expressions, and publication titles.
Boldfaced serif font	Section headings and emphasis.
`Typewriter` font	Syntax of code.
{ }	Alternative items (one or more).
[ ]	Optional items (zero or one).
|	Separator for a list of alternatives.
	Activity.
	Product.
	Product flow.
	Information flow.

Partial examples of work products are presented in the text to illustrate the detailed activity descriptions in Part II, Domain Engineering. These examples are usually excerpts from the Air Traffic Display/Collision Warning Monitor (ATD/CWM) domain case study in Volume II. Rationale, commentary, or advice in support of or to aid understanding and use of an activity description or case study work product are labeled by ***RATIONALE***, ***COMMENT***, or ***NOTE***, either in a separate paragraph or set in parentheses.

Work products examples in this guidebook use a metaprogramming notation to represent *variability* in a product. Variability in a product means that a product will have different content, depending on certain critical *decisions*. A metaprogramming notation allows you to describe how a product's content is determined by those decisions. A simple example of this is the use of a macroprocessor to defer a decision about the size of a data structure. Instead of making the decision on the size of the structure when the code is written (by embedding a constant), you can defer the decision by setting parameters for (parameterizing) the code and supplying the required value at compile time. A metaprogramming notation is an extension of this idea.

Boldfaced, bracketed text is used for metaprogramming notation in this guidebook:

<boldfaced_identifier>	A deferred decision (e.g., **<size>**). Such identifiers may be separated by dots to indicate elements of composite decisions (e.g., **<stack.type>** and **<stack.size>**). This identifier is replaced with the actual value of the decision whenever an instance of a work product is created.

<if predicate **then>** body1 [**<else>** body2] **<endif>** .....
A conditional inclusion. If the predicate evaluates to true, then body1 is included in the work product. If an **else** clause is included and the predicate evaluates to false, then body2 is included in the work product. The predicate is informal and defined in terms of decisions. Also referred to as a conditional term.

<forall ident **in** list**>** body **<endfor>** .....
An iterative (repeating) inclusion. The list is an identifier for a decision that is multi-valued. This construct includes one copy of body in the work product for each value of the decision. For each copy of body included, the corresponding decision value replaces all occurrences of identifier ident in that copy. Also referred to as an iterative term.

A body is any text that may be a part of some work product. A body may also contain nested metaprogramming constructs; if so, those constructs must to be evaluated to determine the content of the body.

OV.2. FUNDAMENTALS OF SYNTHESIS

Synthesis was conceived in recognition of past experience that suggested a need for a major reconception of the software process (Campbell, Faulk, and Weiss 1990) to produce improvements in software productivity, product quality, and customer responsiveness. (Davis, Bersoff, and Comer 1988) present a comparison of several models of software development life cycle, which is a good framework for understanding this need. As the foundation for an initial understanding of Synthesis, this section describes the key principles underlying the Synthesis approach, discusses the context in which Synthesis applies, and provides an overview of how Synthesis works.

2.1 KEY PRINCIPLES

The Synthesis concept depends on four basic principles: *program* families, *iterative processes*, *specifications*, and *abstraction*-based reuse. An understanding of these principles will help you to understand Synthesis.

2.1.1 Program Families

A program family is a set of programs that are sufficiently similar that it is worthwhile to understand the common properties of the set before considering the special properties of individual instances. The concept of program families was first proposed by Dijkstra (Dijkstra 1972) and later elaborated by Parnas (Parnas 1976). Both argue that developers should construct software programs, not as unique artifacts, but as instances of a family of similar programs. A primary distinction of this view is in how the creation of program versions are viewed: not as successive modifications to previous versions, but as rederivations from a common abstraction. Each member of a family can be characterized entirely in terms of how it differs from the common abstraction (i.e., the variations in the family). In Synthesis, this concept of program families is generalized to a concept of product families that encompass all the work products of software development.

2.1.2 Iterative Processes

An iterative process is a process in which completion occurs only after repetition of producing and using activities results in refined work products. In a non-iterative process, an activity continues without interruption until its resulting work products are believed complete. Only after such completion is there any attempt to use those work products in performing other activities. Since many errors in software work products are difficult to discover before detailed use, an iterative process systematically produces better quality results than one that depends on producing correct work products without such repetition. A strong discipline of work product versioning and configuration management is crucial to a successful iterative process.

2.1.3 Specifications

A specification is a complete, precise description of the verifiable properties required of a work product. A specification is an abstract model of a system that aids understanding and analysis. For expressive power, the notation in which a specification is written usually assumes an understanding of specialized terminology and constructs. Normally, a specification is either a description of requirements (i.e., of the problem to be solved) or a description of a design (i.e., of the solution). Additionally in Synthesis, a specification may be a description of a particular problem and its solution in terms of variations that characterize a family of similar problems.

The Synthesis concept of specifications derives from work in both requirements specification and automatic programming. A key objective in both areas is the creation of a notation for describing a required system without unduly constraining the resulting solution. The Naval Research Laboratory Software Cost Reduction project developed an approach to precise, semi-formal specifications of software requirements for avionics systems (Heninger et al. 1978). (Winograd 1979) argue the need for a new view of programming based on a descriptive (i.e., non-prescriptive) language. (Balzer and Goldman 1979) describe criteria for designing and judging the quality of a specification language.

2.1.4 Abstraction-Based Reuse

The direct result of a software development project is a set of work products that describe and implement a software solution to a customer-defined problem. Abstraction-based reuse (Campbell 1989) provides a means for representing a *product family* as a set of adaptable work products and for deriving instances of those work products that represent a particular system. An adaptable work product concisely represents a set of similar work products that vary in well defined ways. Ada generic packages are a notation for representing a set of similar Ada code packages. The TRF2 metaprogramming tool (Software Productivity Consortium 1991b) provides a flexible notation for representing any text-based adaptable work product.

2.2 CONTEXT FOR SYNTHESIS

In initiating a Synthesis practice, you should first consider the context in which you currently work. This context is determined by three concerns: business objectives, system engineering practices, and the objectives of a software engineering process. Within this context, you can create a capability for rapid, systematic delivery of similar, yet varied, systems.

2.2.1 Business Objectives

Synthesis is characterized by its focus on a family of systems for a business area rather than on individual systems. This focus arises from evidence that, within a class of systems, an understanding of similarities provides significant leverage for constructing a great variety of high-quality systems much more cheaply and reliably.

With Synthesis, you conceive a domain not on an objective basis, but as a realization of the declared business objectives of your specific organization. Business objectives determine for an organization the types of systems to be built in the future to meet the needs of customers or other client organizations. A primary consideration in setting these objectives is the expertise already available within your organization, particularly as a result of experience in building systems in the past. Other considerations are your expectations of future customer needs and changing technology.

The environment most conducive to the effective use of Synthesis is one in which you give emphasis to long-term business objectives. A positive climate for investment in the future, possibly at the cost of deferred—but potentially greater—total return is a major advantage for realizing success with Synthesis. It is important to consider these larger business concerns when addressing the needs of a particular customer to avoid arbitarily sacrificing longer-term interests arbitrarily to short-term pressures. On the other hand, Synthesis lends itself to a management philosophy of incremental commitment, both for early pay-back after a minimum initial investment and for accommodating the changing needs of a single customer or the differing needs of many.

2.2.2 System Engineering Practices

Construction of modern, complex systems is a major undertaking that often requires the coordination of many large groups of engineers with expertise in diverse fields. Some of these groups may be organizations in separate companies, working jointly or as subcontractors, to deliver a required system. Engineers follow a discipline of system engineering to partition the problem and apply appropriate expertise to solve each facet of the problem most effectively. This partitioning into subsystems often follows the lines of major hardware components, but it may also serve the purpose of decomposing the system into more manageable software assemblies.

Such partitioning is compatible with Synthesis, particularly when you follow a systematic approach that results in similar partitioning of similar systems. Each software partition (or subsystem) then corresponds to a member of a family of similar subsystems (i.e., software system) that can be the responsibility of a cohesive business organization.

2.2.3 Objectives of a Software Engineering Process

The broad purposes of a software engineering process may be stated simply:

- To allow customers and developers to communicate effectively so that the problem to be solved is understood (i.e., analysis).
- To allow developers to produce a solution that corresponds precisely to the problem as communicated (i.e., synthesis).
- To allow developers and customers to validate the solution as having satisfied the actual problem (i.e., evaluation).
- To allow developers to organize and coordinate their work to perform the process efficiently as a team (i.e., management).

A conventional approach to software development achieves these purposes, but not efficiently. Synthesis is motivated by the need for a systematic approach to tailoring the software engineering process to meet the specific needs of each organization. The key to achieving this end is to consider long-term business needs when developing software. Looking first at the similarities among systems that the organization builds provides a basis for becoming more productive by eliminating redundant effort. Your efforts to build a particular software system can be focused more efficiently on the aspects of the system that are distinctive. Synthesis prescribes the work that is necessary to achieve this potential.

2.3 AN OVERVIEW OF SYNTHESIS

The purpose of Synthesis is to help you better utilize your expertise about a set of similar problems and associated solutions pertinent to your business area. By viewing similar problems as a family, common characteristics provide leverage in building any particular system. Similarities support a form of standardization that enables systematic adaptation to meet the specific needs of a particular customer. The results of standardized decision making by project engineers guide necessary adaptations of standardized, reusable work products.

The primary distinguishing features of Synthesis are:

- Formalization of a domain as a family of systems that share many common features, but that also vary in well-defined ways.
- System building reduced to resolution of requirements and engineering decisions, representing the variations characteristic of a domain.
- Reuse of software artifacts through mechanical adaptation of components to satisfy requirements and engineering decisions.
- Model-based analyses of described systems to help understand the implications of system-building decisions and to evaluate alternatives.

Synthesis defines and integrates two processes: Application Engineering and Domain Engineering. Figure OV.2-1 shows how these processes relate to each other.

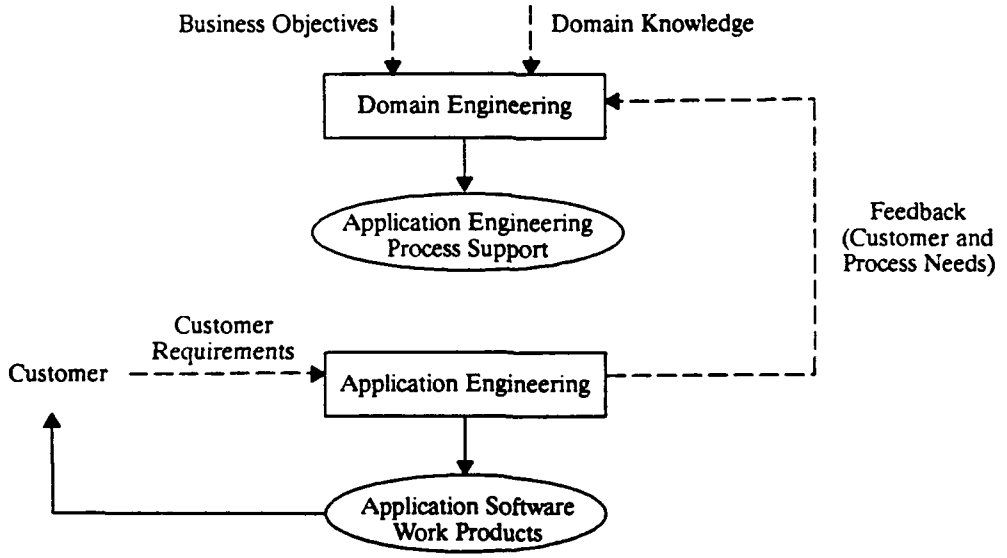


Figure OV.2-1. A Synthesis Process

Application Engineering (described further in Section AE.1) is a stand :dized process by which projects produce and deliver applications to customers. In terms of objectives, this is the equivalent of conventional, project-centered software development. Both start with customer requirements and produce a set of software work products that are meant to satisfy those requirements. Through a process of Domain

Engineering (described further in Section DE.1), however,a simpler, more efficient process of software development may be conceived and supported with standardized, reusable work products.

Rather than a series of analysis, design, implementation, and testing activities, Application Engineering is the creation and validation of a model of a required system. This model is a set of requirements and engineering decisions sufficient to describe a particular system, given a family of such systems. Work products, including code and documentation, are mechanically derived from this model using adaptable forms of those work products provided by Domain Engineering.

Domain Engineering supports Application Engineering in two ways. First, it creates a set of adaptable work products that correspond to these work products that an Application Engineering project must produce. By identifying the decisions that must be deferred until a particular system is needed and parameterizing a work product to show how it varies as a result of those decision, the work product is made adaptable. Second, a standard Application Engineering process is designed that supports the decision making and work-product creation appropriate to projects in the business area. This process is defined as a set of standard procedures and practices, possibly augmented by appropriate automated support.

Both of these processes are iterative since both must support evolving needs. Application Engineering is iterative to accommodate both uncertain and changing customer requirements. Domain Engineering is iterative because changing markets and customer requirements lead to evolving product and process needs of Application Engineering projects.

This page intentionally left blank.

PART II: DOMAIN ENGINEERING

This page intentionally left blank.

DE.1. DOMAIN ENGINEERING OVERVIEW

1. GETTING STARTED

The Domain Engineering Activity is an activity of Synthesis that creates and supports a standardized Application Engineering process and products in a business area. Domain Engineering is a comprehensive life-cycle process that provides the management, analysis, implementation, and support of a product family of primary value to a business-area organization.

1.1 Objectives

The objectives of Domain Engineering are:

- To organize and direct resources to accomplish the business objectives of an organization.
- To define the nature, extent, and substance of a product family that fulfills those business objectives.
- To provide leverage by which projects charged with delivering a product within the bounds of the domain can do so more effectively, predictably, and profitably.

1.2 Required Information

Domain Engineering requires the following information:

- Domain knowledge
- Business objectives

1.3 Experience and Expertise Needs

Domain Engineering requires the following expertise:

- Understanding of the needs that motivate systems.
- Understanding of the environments in which the systems are to operate.
- Understanding of how the systems are built.

2. PRODUCT DESCRIPTION

Domain Engineering creates four products:

- *Domain Plan* (see Section DE.2), which describes a plan for *domain evolution* and defines the tasks and resource allocations for domain development increments.

- *Domain Definition* (see Section DE.3), which defines the informal scope and orientation that characterize a viable domain.

- *Domain Specification* (see Section DE.3), which formalizes expert knowledge of how to express problems in the domain and how to create corresponding solutions for those problems.

- *Domain Implementation* (or Application Engineering Process Support) (see Section DE.4), which defines in detail (with documentation and automated support) the process and products of Application Engineering in the domain, as prescribed by the Domain Specification.

3. PROCESS DESCRIPTION

The Domain Engineering process (shown in Figure DE.1-1) is an interaction among four subactivities:

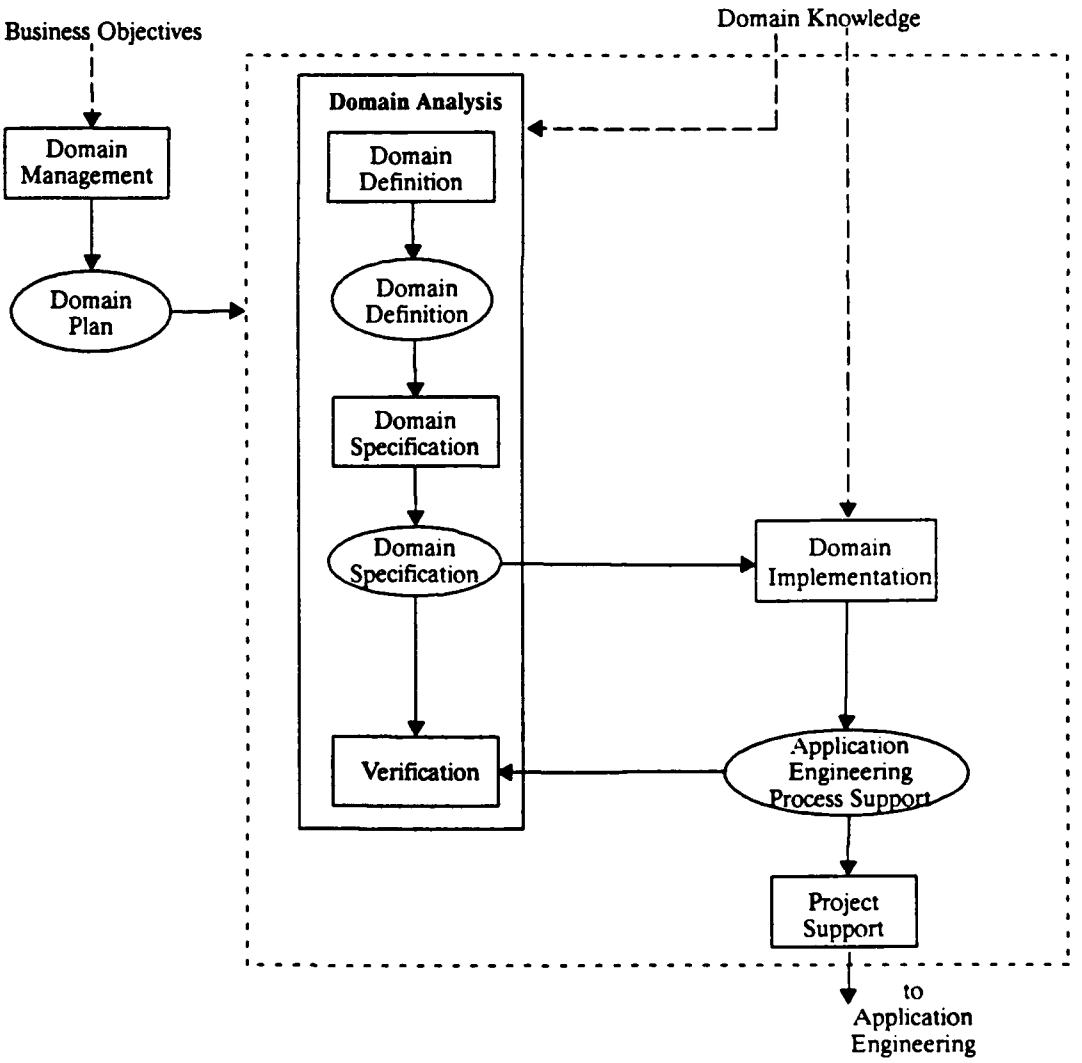


Figure DE.1-1. Domain Engineering Process

- Domain Management (see Section DE.2), an activity for the planning, monitoring, and controlling of domain resources to achieve organizational business objectives.
- Domain Analysis (see Section DE.3), an activity for the study and formalization of domain knowledge to standardize the products and process of Application Engineering.
- Domain Implementation (see Section DE.4), an activity for the creation of standards and procedures, in documentation and supporting automation, that institute standardized products and process of Application Engineering in accordance with a Domain Specification.
- Project Support (see Section DE.5), an activity for delivery of Application Engineering Process Support to projects, support of its use, and evaluation of its effectiveness.

Risk Management

The following are the risks and possible responses in the Domain Engineering Activity:

Risk The products of Domain Engineering (standardization of Application Engineering products and process) will not lead to standardized domain practices on projects.

Implication The Domain Engineering investment will not produce projected benefits for the business organization.

Mitigation

- Staff the Domain Engineering work with experienced project managers and engineers. Ensure that all work is actively reviewed by other experienced managers and engineers and is adequately reviewed by all participants of Application Engineering projects.
- Evaluate the Domain Engineering process and products relative to past project experiences. Ensure that the characteristics of that experience or the resulting systems are not inadvertently in conflict with the process and products.
- Provide unified management of Domain Engineering and Application Engineering projects. Establish an organizational commitment to the combined success of all participants.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback To Information Sources

4.2 Feedback from Product Consumers

Contingency The standardized product family is inadequate to support the needs of particular customers.

Source Application Engineering.

Response

- Determine that expressed needs are outside of or otherwise conflict with chosen business objectives or are not viable to satisfy given available domain resources.
- Evolve the domain to accommodate changing needs.

Contingency The standardized Application Engineering process is inefficient or leads to less-than-ideal results for a particular project.

Source Application Engineering.

Response

- Determine that the benefits of process standardization outweigh the interests of the particular project.
- Evolve the definition of the Application Engineering process to reflect this project's experience or to be adaptable to the particular conditions of concern.

DE.2. DOMAIN MANAGEMENT ACTIVITY

1. GETTING STARTED

Domain Management is an activity of Domain Engineering for the management of business-area resources to achieve assigned business objectives. The business-area organization provides resources and direction for both Domain Engineering and associated Application Engineering projects. Strategic planning determines the nature of the market addressed and how resources are allocated between Domain Engineering and Application Engineering projects. Tactical planning determines how Domain Engineering resources are applied to create an efficient Application Engineering process and a high-quality product family. The activities of both Domain Engineering and Application Engineering are monitored to assess progress, ensure proper adherence to plans, and guide necessary revisions in them. A key concern of Domain Management is the coordination of Domain Engineering activities to satisfy the needs and priorities of client Application Engineering projects to achieve in turn the overall objectives of the business area.

1.1 Objectives

The objective of Domain Management is to manage business-area resources to achieve an assigned strategic mission. Management establishes objectives for the organization that guide the development of strategic (domain evolution) and tactical (domain development) plans. Domain evolution is concerned with the overall trends in the market for the business area and how resources should be applied to serve best the evolving market. The primary element of domain evolution planning is the identification of a series of domain-development increments that are projected to accomplish strategic objectives most efficiently. For each of these increments, you develop a plan to guide the activities of Domain Engineering in supporting the current needs of client Application Engineering projects. Each Application Engineering project is planned independently to meet particular customer needs consonant with business-area objectives and priorities (i.e., with an awareness of domain capabilities).

1.2 Required Information

The Domain Management Activity requires on business objectives.

1.3 Expertise Needs

The Domain Management Activity requires the following expertise:

- Experience in the creation of long-range business plans.
- An understanding of the business and the market characteristics of the business area.

- Experience in all aspects of business-area management.
- Experience in all aspects of product-oriented project management.

2. PRODUCT DESCRIPTION

Name Domain Plan.

Purpose Define long-range and near-term objectives and organize and manage domain resources to achieve those objectives.

Content The Domain Plan consists of two parts:

- Domain Evolution Plan.
- Domain Development Plan.

These parts are described in Sections 2.1 and 2.2, respectively.

Verification Criteria There are two verification criteria for the Domain Plan:

- The plan must show how near-term objectives contribute to long-range objectives. All near-term objectives must support long-range objectives.
- The projected market for products in the business area must be large enough to compensate sufficiently for projected costs and risks.

2.1 Domain Evolution Plan

Purpose The Domain Evolution Plan defines long-range objectives for the domain and organizes resources to achieve them. This plan is strategic in nature and recognizes that both an organization's capabilities and its opportunities for profitable business change over time. Not all objectives can be met initially but must develop in the course of time, in increments that balance alternative uses of available resources against the potential for return.

Content The Domain Evolution Plan is a long-range business plan for the domain. It must set forth a strategy for achieving business objectives. To be effective, a Domain Evolution Plan must address the following questions:

- What are the critical aspects of your market?
- Who are your customers and what factors are most important to them in deciding to award contracts?
- What type of products will you produce to address this market?

- How many units do you expect to produce both in the first year and as the domain matures?
- Who are your competitors? What are their strengths and weaknesses?
- What are your strengths and weaknesses?
- How do you expect the market, your business, and your competition to change over time?
- How are resources to be allocated between domain development and Application Engineering projects?
- What degree of automation do you expect to deploy in support of Application Engineering, both initially and as the domain matures?
- What are the long-term risks in developing the domain, and how will you mitigate each one?
- How are standards of consistency and quality to be achieved in domain practices?
- What are the objectives for the domain, and what are the objectives of each increment of development that is to achieve those objectives?

Form and Structure To the extent possible, the form of a Domain Evolution Plan should be the form your organization currently uses for long-range business plans.

Verification Criteria

- The plan is complete (i.e., it addresses all business objectives).
- The plan is credible (i.e., it sets forth a strategy that is feasible, given projected resources).
- *Domain objectives* are realistic, given the p cted availability of resources and strength of the competition.
- All important risks are identified and a sed.
- Criteria are defined to measure progr against objectives, and to choose among alternative plans or indicate a n for replanning.

2.2 Dom Development Plan

Purpose The Domain Development Plan specifies how Domain Engineering resources are to be organized and managed to achieve near-term domain objectives.

Content The Domain Development Plan includes the following:

- Risk analysis. Identification of uncertainties in meeting domain development objectives and assessment of the risks of failure.
- Near-term domain objectives that reflect long-range domain objectives as well as needs of current projects. Objectives are divided into risk objectives and product objectives. Risk objectives attempt to mitigate risks identified in the risk analysis. Product objectives attempt to create specific products.
- Schedule. The allocation of domain resources to activities that satisfy objectives. The schedule establishes specific milestones and success criteria for the other activities of Domain Engineering.

Form and Structure To the extent possible, the form of a Domain Development Plan should be the form your organization uses for project planning.

Verification Criteria The Domain Development Plan institutes a plan that seems likely to achieve the objectives assigned to the Domain Engineering increment by the Domain Evolution Plan.

3. PROCESS DESCRIPTION

The Domain Management Activity consists of two subactivities: the Domain Evolution Plan Activity and the Domain Development Plan Activity. These activities are performed as depicted in Figure DE.2-1.

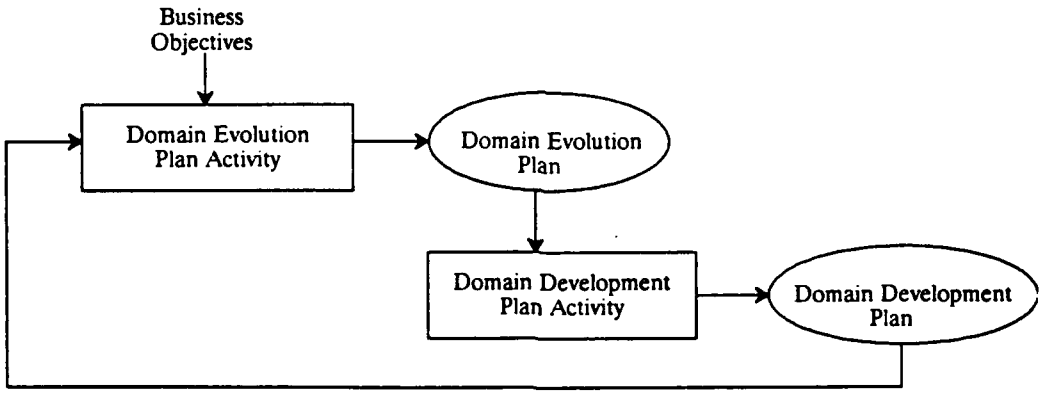


Figure DE.2-1. Domain Management Process

The Domain Evolution Activity is first performed upon creation of a domain and again after each completion of a Domain Development Plan. Iteration terminates when the domain is no longer judged to be economically viable.

The Domain Development Activity is performed iteratively. The time scheduled for a particular iteration depends on the objectives defined in the Domain Evolution Plan for that iteration.

3.1 Domain Evolution Plan Activity

3.1.1 Procedure

These steps are followed for the domain evolution plan activity.

Step 1. Set Domain Objectives

Action Develop a set of domain objectives that will be used to guide the long-term evolution of the domain.

Input Business objectives.

Result Domain objectives.

Heuristics

- Develop a preliminary statement of objectives that expresses the strategic mission of the organization.
- Develop market projections as a basis for understanding current and future customer needs to be met. Refine the objectives to match perceived market opportunities.

Step 2. Develop Practices and Procedures

Action Develop and document standard practices and procedures to be followed in the activities of Domain Engineering.

Input Current practices.

Result Practices and procedures.

Heuristics

- Prescribed practices and procedures should encompass administrative, software development (e.g., requirements and design methods, coding and documentation standards), project management and control, and quality assurance (e.g., testing, walkthrough, and review procedures).
- Configuration management procedures are a key element for controlling iterative domain development. Each iteration of domain development is represented by one version of each Domain Engineering work product that you produce. Feedback on the use of a product version leads to the creation of a new version in a later iteration of development.

Step 3. Plan Domain Evolution

Action Create a plan for Domain Evolution.

Input Domain objectives.

Result Domain Evolution Plan.

Heuristics

- Describe a strategy for achieving domain objectives. The life cycle of a domain passes through four phases:
  - Conception: A small, cohesive group explores the boundaries of a domain to establish a viable market basis; project support is not viable yet.
  - Elaboration: One or a few very similar projects are directly supported; domain evolution emphasizes needs that are important to most customers.
  - Expansion: Supporting the planned diversity of projects is now viable; market opportunities drive further domain evolution.
  - Consolidation: Little additional diversification is viable; projects are managed to fit within the supported/variations in domain capabilities as much as possible.
  - Develop a profile of the market to be supported as the domain matures. Provide details in terms of evolving domain objectives.
  - Develop a profile of the evolution of automation to be developed in support of Application Engineering projects.
  - Develop a profile of the resources needed to fulfill domain objectives. Consider both quantitative and qualitative needs in all skill categories. Project how these resources are to be allocated between Domain Engineering and Application Engineering projects.
  - Define a series of domain development increments, and define objectives, consistent with domain objectives and allocate them to increments.

3.1.2 Risk Management

Risk Domain plans will not be met within schedule with allocated resources.

Implication Domain capabilities will fall short of plans.

Mitigation

- Review plans with experienced engineers to ensure that planned development is technically viable.
- Reevaluate domain objectives and replan domain evolution to provide for shorter iterations that achieve essential capabilities sooner; defer less important objectives.

Risk Market needs will not be met by projected development.

Implication Demand for projects will not be sufficient to justify planned investment in the domain.

Mitigation Review objectives and plans with marketing and major customers to ensure that market needs are properly represented.

3.2 Domain Development Plan Activity

3.2.1 Procedure

The process for the Domain Development Plan Activity is shown in Figure DE.2-2. This description assumes you are experienced in the practices of project management. Alternatively, the project management section of the *Evolutionary Spiral Process Guidebook* (Software Productivity Consortium 1991c) presents a detailed description of a project management method that you can follow to elaborate this procedure.

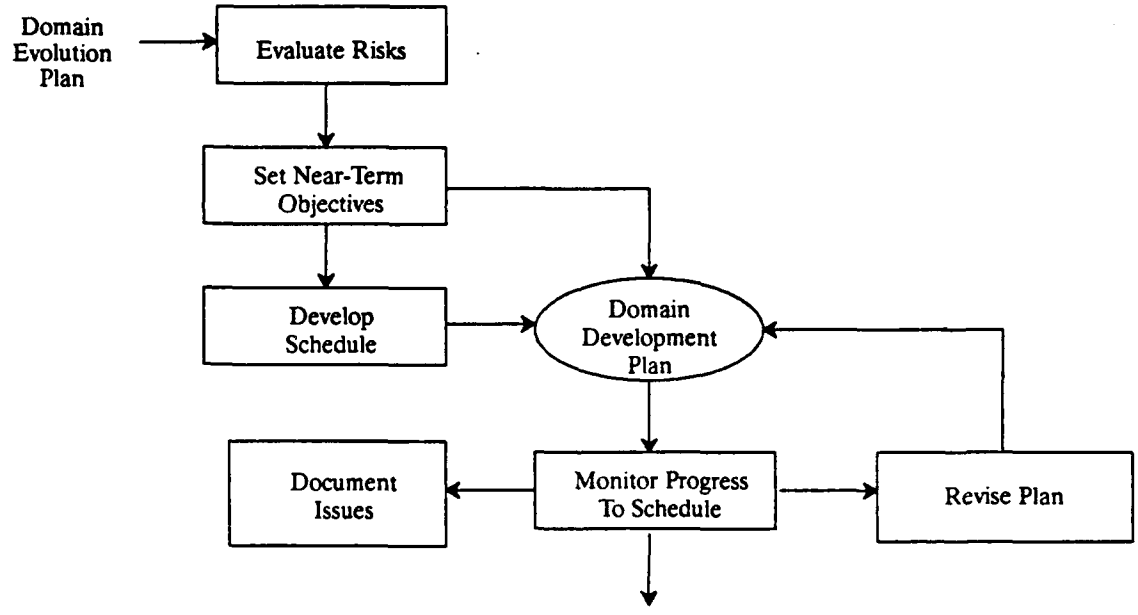


Figure DE.2-2. Domain Development Plan Activity

These steps are followed for the domain development plan activity.

Step 1. Evaluate Risks

Action Identify and rank the domain development risks faced by the organization.

Input

- Domain Evolution Plan.
- Requests from client application projects and previous domain development experience.

Result Development Risk Analysis.

Heuristics

- A recurring class of risks that domain development must face is the ability to create the products that will be required by client application projects in the near term. Another recurring class of risks involves how to evolve the domain to meet long-range domain objectives. Actions to mitigate these risk classes may conflict.
- Another important class of risks relates to problems discovered in previous iterations of the Domain Engineering process.

Step 2. Set Near-term Objectives

Action Develop a prioritized set of near-term objectives that address the risks identified in the Development Risk Analysis.

Input

- Domain Evolution Plan.
- Development Risk Analysis.

Result Development Objectives.

Heuristics

- A domain-development approach must balance the long-term Domain Objectives against the short-term client project needs.
- Each objective should have associated success criteria that are used to determine if the objective has been met. The success criteria should be written in such a way that they are directly measurable (if possible).
- Objectives are often stated in terms of variations that characterize systems in the domain or variations to be allowed in the practice of Application Engineering.

Step 3. Develop Schedule

Action Develop a schedule that allocates resources to tasks.

Input Development Objectives.

Result Domain Development Plan.

Heuristics

- Create specific goals and completion criteria for each task. Each task is characterized by a Domain Engineering activity to be performed and completion criteria appropriate to that activity. The full set of tasks must address the near-term objectives within the resource budget provided. If the resource budget does not allow all the objectives to be addressed, the objectives with the highest priority should be addressed.

- Plan for short iterations so that mistakes made in front-end tasks may be caught and corrected quickly in a subsequent iteration. Iterations at the beginning of the life cycle of a domain should be particularly short (three to four months) to compensate for the likely learning curve in domain concepts.

Step 4. Monitor Progress to Schedule

Action Monitor Domain Engineering work progress to planned milestones and completion criteria.

Input Domain Development Plan.

Result

Heuristics The schedule establishes milestones and completion criteria for activities that are used to evaluate progress. Whenever new issues are identified or progress differs from that planned, evaluate whether to document your concerns for future planning or rather to revise the current plan for immediate action.

Step 5. Document Issues

Action Create a record of any issue that arises in the performance of the plan.

Input

Result Feedback to the Domain Evolution Plan Activity.

Heuristics Document the source, implications, and possible and actual resolutions of each issue.

Step 6. Revise Plan

Action Modify the Domain Development Plan to correct for greater or lesser success than planned.

Input Domain Development Plan.

Result Revised Domain Development Plan.

Heuristics Revise the plan to add or defer activities or to reallocate time and resources among planned activities, as priorities dictate.

3.2.2 Risk Management

Risk Employees resist using standardized practices and procedures.

Implication Inefficient operation and employee dissatisfaction will reduce productivity.

Mitigation

- Involve employees in development of practices and procedures.
- Provide education and apprenticeships.
- Conduct pilot projects that emphasize learning new skills over product delivery.

Risk Projects fail to recognize when to terminate an iteration.

Implication

- This excessive detail in products without adequate foundation or potential benefit.
- Schedules will slip.

Mitigation Review objectives and completion criteria to make sure they are specific and understood by the domain engineers.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Domain Definition and/or Domain Specification fails to provide the needed capabilities required by the Domain Plan.

Source Domain Analysis Activity.

Response Describe ways in which the Domain Definition and/or Domain Specification fail to provide the necessary capabilities. Modify schedule to allow completion of indicated Domain Definition or Domain Specification revisions.

4.2 Feedback from Product Consumers

Contingency Customer needs are not being met by the domain.

Source Application Engineering.

Response

- Revise the Domain Plan to accommodate new needs.
- Determine that needs are incompatible with your organization's business objectives.

Contingency Practices and procedures are either ineffective or inefficient.

Source
- Domain Analysis Activity.
- Domain Implementation Activity.
- Project Support Activity.

Response Revise practices and procedures to reflect domain experience.

Contingency The Domain Plan is too ambitious for available resources or expertise.

Source
- Domain Analysis Activity.
- Domain Implementation Activity.

Response
- Allocate additional resources or time to domain development.
- Refine the Domain Plan to reduce the scope.

This page intentionally left blank.

DE.3. DOMAIN ANALYSIS ACTIVITY

1. GETTING STARTED

Domain Analysis is an activity of Domain Engineering for the study and formalization of a business area as a domain. The purpose of formalizing a business area is to standardize and leverage knowledge of how recurring and varying customer requirements affect the form and content of a product. The scope of a domain is a business decision based on evaluations of available expertise and potential business opportunities. Domain Analysis specifies a standardized Application Engineering process and product family, and verifies that a corresponding Domain Implementation meets that specification.

1.1 Objectives

The objectives of Domain Analysis are:

- To determine scope and to evaluate the economic *viability* of a domain.
- To establish a managed and evolving repository of knowledge about the domain.
- To produce a specification for an Application Engineering process and product family appropriate to the domain.

1.2 Required Information

Domain Analysis requires the following information:

- Domain knowledge
- Business area knowledge
- Domain Plan

1.3 Experience and Expertise Needs

Domain Analysis requires the following expertise:

- Understanding of the needs that motivate these systems.
- Understanding of the environments in which these systems operate.
- Understanding of how these systems are built.

2. PRODUCT DESCRIPTION

Domain Analysis creates two work products:

- *Domain Definition* (see Section DE.3.1).
- Domain Specification (see Section DE.3.2).

3. PROCESS DESCRIPTION

The Domain Analysis process is shown in Figure DE.3-1.

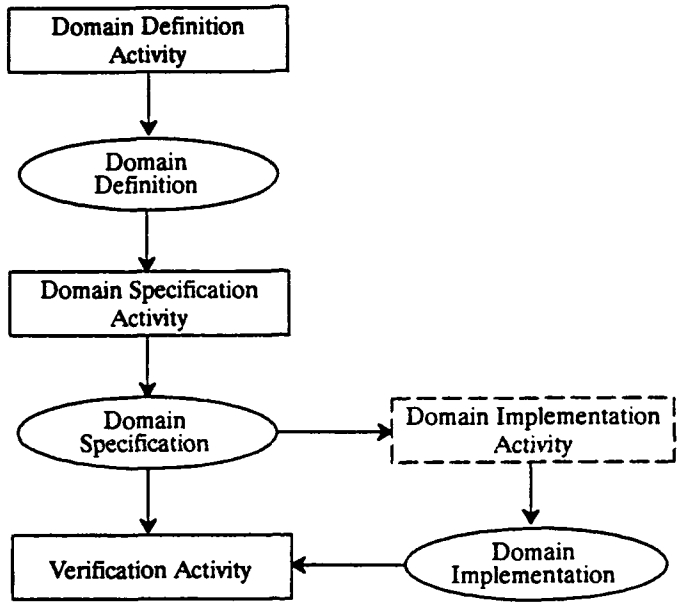


Figure DE.3-1. Domain Analysis Activity Process

These steps are followed for the domain analysis activity.

Step 1. Domain Definition Activity

Action Characterize the domain to satisfy domain objectives (see Section DE.3.1).

Result Domain Definition.

Heuristics

- Characterize the domain by defining its scope (i.e., classes of systems included in and excluded from the domain) and how included systems are distinguished from one another. These definitions are a basis for judging the qualitative and economic characteristics of the domain to determine whether the domain as defined will be economically viable.
- If analysis of the Domain Definition fails the test of economic viability, reevaluate the scope of the domain in terms of domain objectives.

Step 2. Domain Specification Activity

Action Specify Application Engineering Process Support (see Section DE.3.2).

Result Domain Specification

Heuristics

- Create a standard Application Engineering process for the domain. Ensure that all needs of projects are flexibly supported.
- Design an *Application Modeling Notation* for communication of system requirements and *constraints* among customers and application engineers. Identify the decisions that an application engineer must make to describe fully the variations in a system fully. This notation must accommodate both functional (e.g., behavioral) and nonfunctional aspects (e.g., size, timing, fault tolerance, hardware architecture, hardware/software configuration) so that the application engineer can express customer requirements adequately. This notation should be based on existing (formal or informal) notations used by domain experts.
- Ensure that the Application Modeling Notation is precise enough to be used as a source for mapping into exact system solutions. Create standardized requirements for the domain. This description must establish both the common and variable aspects of the behavior and constraints of product family members. An unambiguous specification of requirements is needed so that domain implementors can determine what impact a decision has on a system. This also provides a basis for explaining the notation to application engineers.
- Create a standardized design for the domain. Such a design must satisfy both the common and variable aspects of the product family. A standardized design includes both *design structures* that define various views of product structure and components from which a product is constructed in order to satisfy customer requirements.

Step 3. Verification Activity

Action Verify that the Domain Implementation satisfies the Domain Specification (see Section DE.3.3 in future releases).

Risk Management

Risk The cost of an increment of Domain Analysis is projected to exceed the budget.

Implication Insufficient resources exist to complete a planned iteration of Domain Engineering.

Mitigation

1. Reduce the current scope.
2. Seek a change in domain objectives or an increase in the budget for the increment from Domain Management.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback To Information Sources

Contingency The Domain Plan cannot be satisfied with available technical capabilities.

Source Domain Management.

Response Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete Domain Definition and a Domain Specification that satisfy the Domain Plan as closely as possible.

Contingency The Domain Implementation does not satisfy the Domain Specification.

Source Domain Implementation.

Response Clarify the intent of the the Domain Specification.

Contingency The practices and procedures specified in the Domain Plan are either ineffective or inefficient.

Source Domain Management.

Response Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

4.2 Feedback from Product Consumers

Contingency Suggestions are made for Domain Specification changes to exploit unforeseen opportunities. For example, a situation where substantial software is made available for use in the Domain Implementation that was not available when the Domain Specification was [illegible].

Source Domain Implementation.

Response

1. Revise the Domain Specification.
2. Refer to Domain Management for future planning.
3. Reject the changes due to conflicts with the Domain Definition.

Contingency The Domain Definition and/or Domain Specification fails to provide the needed capabilities required by the Domain Plan.

Source Domain Management.

Response Evolve the Domain Definition and the Domain Specification to be consistent with the Domain Plan.

Contingency The Domain Specification is incomplete, ambiguous, or inconsistent.

Source Domain Implementation.

Response Revise the Domain Specification to correct the inadequacies.

Contingency The standardized Application Engineering process is inefficient or leads to less-than-ideal results for a particular project.

Source Application Engineering.

Response

1. Determine that the benefits of process standardization outweigh the interests of the particular project.
2. Evolve the definition of Application Engineering process to reflect this project's experience or to be adapted to the particular conditions of concern.

This page intentionally left blank.

DE.3.1. DOMAIN DEFINITION ACTIVITY

1. GETTING STARTED

Domain Definition is an activity of Domain Analysis for creating a Domain Definition. A Domain Definition is an informal description of the systems in the business area that form a domain. A Domain Definition characterizes how systems in the domain are similar and how they differ.

1.1 Objectives

The objectives of the Domain Definition Activity are:

- To establish a conceptual basis and bounds for more detailed Domain Analysis.
- To determine whether planned development and evolution of the domain is viable.
- To establish criteria by which management can judge whether a proposed system is properly within the domain.

1.2 Required Information

The Domain Definition Activity requires the following information:

- Domain knowledge
- Descriptions of similar domains
- Business area knowledge
- Domain Plan

1.3 Expertise Needs

The Domain Definition should be developed by people with a variety of backgrounds in the business area under study, including engineering, business, and management experience. Specific expertise is needed for:

- Broad *business-area knowledge*: What systems have been built; the nature of systems likely to be requested and built; new features and technology that will impact on customer expectations.
- Broad market awareness: What contracts are forthcoming; what kind of competition there will be for those contracts; the long-term growth potential of this business area.

- Proposal development experience: What aspects of a system are critical to winning a contract proposal in this business area.

- System development and management experience: Size, cost, and schedule estimation for systems in this business area; an intuitive understanding of the technical difficulty and *feasibility* of a given feature; knowledge of the common areas of confusion or vagueness in requirements or high-level design; knowledge of the common variations or changes to systems in this business area over the entire life cycle, including development, installation, and maintenance.

2. PRODUCT DESCRIPTION

Name Domain Definition.

Purpose A Domain Definition establishes the scope of a domain and a justification of its economic viability. It provides a basis for determining informally whether a system is properly within that scope.

The definition does not answer detailed questions of scope, but clearly includes and excludes broad classes of systems. Assumptions of *commonality* and exclusion identify the common features of systems in the domain, thereby establishing a family. Assumptions of variability identify how systems in the family are then distinguished from one another. Justification provides a basis for judging qualitative and economic characteristics of the domain to evaluate whether there is sufficient confidence in the economic viability of developing the business area as a domain.

Content A Domain Definition consists of the following components:

- *Domain Synopsis*: An informal statement of the scope of the domain.
- *Domain Glossary*: Definitions of significant terminology used by experts in discussing requirements and systems in the domain.
- *Domain Assumptions*: A description of what is common to all systems in the domain, what is excluded from systems in the domain, and what varies among systems in the domain.
- *Domain Status*: An assessment of the current maturity of the domain relative to its planned evolution.
  - *Viability Analysis*: An analysis of economic viability for the domain over its life cycle.

Verification Criteria

- Every Domain Assumption must be endorsed for every customer type in the customer base identified in a Viability Analysis.
- The needs of all customer types in the customer base must be fully met by the Domain Synopsis and Domain Assumptions.

2.1 Domain Synopsis

Purpose The Domain Synopsis is an informal statement of the scope of the domain. It characterizes systems included in the domain.

Content A Domain Synopsis includes an informal characterization of the systems that make up the domain.

Form and Structure A Domain Synopsis is a simple narrative using terms defined in the Domain Glossary.

The Domain Definition in Volume 2, Section CS.1.4 includes a Domain Synopsis for the ATD/CWM Domain.

Verification Criteria

- The Domain Synopsis must give an intuitive feel for the definitive characteristics of systems in the domain. It should in itself adequately describe any existing or potential system.
- A term that could have different meanings to different readers may be used in the Domain Synopsis only if it is defined in the Domain Glossary.

2.2 Domain Glossary

Purpose The Domain Glossary is a compendium of precise definitions for all significant terminology used by experts for discussing problems (requirements) and solutions (systems) in the domain. This domain terminology is organized into a taxonomy of terms.

Content A Domain Glossary includes a set of standard terms and their definitions.

Form and Structure A Domain Glossary has three parts:

- References to standard, identified glossaries.
- Definitions.

  Comment: Terms are defined in alphabetical order using the following forms:

Term 1	Definition (source of term).
Term 2	(1) First definition (source). (2) Second definition (source).

Form and Structure

- Hierarchic relationships among terms:
  - Specializations of terms (a taxonomy). Every term that is thought of as a categorization of things is included in this structure, along with any terms that denote subcategories. This hierarchical structure is presented in a graphic or indented list form.
  - Composite terms. Every term that is thought of as descriptive of something with parts is included in this structure, along with the terms that denote those parts. This hierarchical structure is presented in a graphic or indented list form.

The Domain Definition in Volume 2, Section CS.1.4 includes a Domain Glossary for the ATD/CWM Domain.

Verification Criteria

- The Domain Glossary must contain precise definitions of all significant terminology used by domain experts for discussing the requirements or engineering of systems in the domain.
- Any term used in a definition that could have different meanings to different readers must also be defined.
- All independently-used terms that are generalizations, specializations, or components of defined terms must also be explicitly defined.
- Terms defined in the Glossary must be sufficient for a domain expert to give an accurate description of any existing or potential system.
- Any term that occurs in a hierarchic relationship must be defined.

2.3 Domain Assumptions

Purpose

Domain Assumptions describe what is common to all systems in the domain and in what significant ways those systems vary and can be distinguished. These determine informally whether a system is within the scope of the domain.

Content

- Commonality Assumptions: A set of assumptions about the characteristics that are common to all systems in the domain (commonalities).
- Variability Assumptions: A set of assumptions about the characteristics that distinguish systems in the domain (variabilities).
- Exclusionary Assumptions: A set of assumptions about the characteristics of systems that are outside the scope of the domain (exclusions).

Form and Structure

Domain Assumptions are comprised of three parts:

- A description of each assumption.
- An informal justification for each assumption.
- A grouping of assumptions around Glossary terms to show that the set of assumptions is complete and to correlate assumptions to particular characteristics of the domain.

The Domain Definition in Volume 2, Section CS.1.4 includes Domain Assumptions for the ATD/CWM Domain.

Verification Criteria

Commonality and Variability Assumptions must capture all important aspects that are common to all systems in the domain and the significant ways in which these systems can vary. Exclusionary Assumptions must not exclude needed capabilities.

- Every Commonality Assumption must apply equally well, without qualification, to any system in the domain. Systems must not violate a stated commonality, either by excluding an included feature or by including an excluded feature.
- Systems must not vary in a way not implied by the set of Variability Assumptions.
- All reviewers must agree that domain experts will consider in Domain Assumptions to be consistent and unambiguous, relative to the definitions in the Domain Glossary. A term that could have different meanings to different readers may be used in a Domain Assumption only if it is defined in the Domain Glossary.

2.4 Domain Status

Purpose

Domain Status describes the current technical maturity of the domain relative to planned evolution and assesses the viability of planned evolution. Of particular concern are unsupported Variability Assumptions (i.e., default commonalities).

The Viability Analysis describes evaluations that establish whether the domain, as defined, will be economically and technologically viable. Qualitative and quantitative criteria assess whether current development and future evolution of the domain will support the organization's business objectives.

Content

The Domain Status is an informal characterization of the degree to which Domain Objectives are satisfied by past development. It includes a Viability Analysis of that status and the effects of further planned development.

Content At a minimum, the Viability Analysis must include:

- An endorsement that the Domain Synopsis and Domain Assumptions define economically viable domain.
- A concise statement that characterizes the confidence (or risk) associated with this endorsement. If possible, this should include a justification of the endorsement and a list of major unresolved issues or risks that may jeopardize the domain's viability.

Depending on the size of the potential business area and the attendant commitment to the domain, however, you may need a more detailed Viability Analysis consisting of some or all of the following:

- The target customer base. This is a profile of each type of customer and/or contract that comprises this area of business. Each profile on the list consists of:
  - The name of the customer (or contract) type.
  - A short description.
  - A list of attributes that characterize this type of customer. These attributes fall into two categories: technical expectations and administrative expectations. Technical expectations are features of the delivered products or the process for developing, testing, maintaining, or delivering and installing the products. Administrative expectations are gross cost, profit, and schedule aspects of a contract.
  - A list of specific (potential or current) contracts and/or customers that fall under this type.
- A grouping of the different alternatives of Variability Assumptions into priority subsets. Minimally, there are two subsets: "must have" and "nice to have." A system of numerical weighting may be desirable.
- A statement of the potential value of the domain. You may need to analyze alternative scopes of the domain separately. The following factors express the potential value of a particular scope:
  - The size of the potential market (consisting of the target customer base profiled above).
  - The planned share of that market.
  - The projected income from that share.

- The expected cost of supporting that market.
- An expected profit margin to justify the risk of entering the market.
- A pricing structure for systems to be produced that supports the cost/profit expectations above.

Form and Structure No established standard form exists for a Domain Status or its Viability Analysis, especially in the case of minimum content. The maturity of a domain can generally be expressed however, as limitations in satisfying Variability Assumptions. Characaterize the technical expectations of the target customer base as the subset of possible alternatives of Variability Assumptions that are supported. Risks can be mitigated by imposing limits on Variability Assumptions.

Verification Criteria

- Every alternative of every domain Variability Assumption must have an explicit (or overall group) economic justification from the potential value analysis and/or an endorsement from the customer base.
- The price/capability of systems offered (as supported by the domain value analysis for a particular domain alternative) must compare sufficiently well against the perceived capabilities of the competition in this business area to justify the anticipated profit and market share.

3. PROCESS DESCRIPTION

3.1 Procedure

These steps are followed for the Domain Definition Activity:

Step 1. Define the Domain Informally

Action Create a description of the domain, characterizing key technical objectives of included systems.

Input Domain Objectives.

Result Domain Synopsis.

Heuristics

- Start with a one-sentence description of the family of systems that constitutes the domain.
- Refine the Domain Synopsis to two pages, at most, of intuitive and not overly-restrictive text. Impart concisely, an informal but complete sense of the domain in the first paragraph. Try to focus on the essential nature, scope, and variety of systems in the domain.

- Include a characterization of the type of problem that systems in the domain solve, a characterization of the external environment (i.e., devices, systems, and users) with which systems interact, and a description of the observable behavior that systems exhibit in solving the problem. You might also establish significant constraints concerning how the systems operate in terms of performance, reliability, or distribution concerns.

- Cover the primary functions performed by every system in the domain and any important functions performed by only some systems. Maintain a "black-box" perspective when describing functional aspects of the system.

- Use terms defined in the Domain Glossary to keep the Domain Synopsis short.

- If the domain (e.g., process control systems) is based on formal theories that provide experts with a common language of communication about problems, refer to those theories in the Domain Synopsis.

Step 2. Establish Standard Terminology

Action Create definitions of all significant terms used by domain experts in discussing the requirements or engineering of systems in the domain.

Input Domain Synopsis.

Result Domain Glossary.

Heuristics

- Maintain term definitions in alphabetical order for ease of reference. Provide cross references to related terms.

- Use definitions from standard glossaries where possible. Make note of such sources in each definition for future traceability.

- Make definitions as precise as possible.

- Make sure that all terminology used in the Domain Synopsis is defined in the Glossary.

- Create a structure that shows term specializations and relationships among similar concepts. This action will reveal missing terms that represent generalizations or specializations of known terms.

- Create a structure that shows the composition of terms and the interrelationship of independent concepts in the formation of logical structures. This will reveal missing terms that are necessary to complete the definition of other terms or terms that tie other terms together into more complex concepts.

Step 3. Establish Domain Assumptions

Action Create lists of the assumptions that allow you to think of the envisioned set of systems as a family and the assumptions that allow you then to distinguish among them.

Input

- Domain Synopsis.
- Domain Glossary.

Result Domain Assumptions.

Heuristics

- State only those assumptions that affect the system software and associated delivered products (e.g., documentation, test support, etc.).
- To create a preliminary set of assumptions:
  a. Create a Commonality Assumption for each characteristic specified in the Domain Synopsis that is shared by all systems in the domain.
  b. Create a Variability Assumption for each characteristic specified in the Domain Synopsis that is not shared by all systems in the domain.
  c. For each term in the Glossary, determine whether the term indicates a commonality or a variability among systems in the domain. Create an Assumption accordingly.
- Make Variability Assumptions precise by indicating the type of decision the application engineer must make to resolve the variability. It is not sufficient to note only that some characteristic varies. It is important to establish how much flexibility the application engineer needs to characterize different systems adequately.
- Compare the characteristics of existing systems to facilitate the identification of common features and variations.
- Consider characteristics anticipated for future systems to identify additional Variability Assumptions.
- Look at the maintenance history of existing systems for an indication of how systems in the domain change over their life cycles. The histories of these systems indicate likely variabilities that characterize the evolution of individual systems.
- Use information on technological advancements that could impact the development of future systems in the domain to identify potential variations.
- Distinguish between system generation time and run-time variations when developing assumptions about variable aspects of the domain. Treat a run-time variation that is characteristic of all systems in the domain as a commonality.

- Use Exclusionary Assumptions to clarify a domain's boundary. The intention should be not to enumerate every type of system or function that is outside the domain, but rather to exclude explicitly those functions or characteristics that might reasonably come to mind when reading the Domain Synopsis. Thus, you can answer the question of whether a particular system belongs within the domain more directly by checking the exclusions. Exclusions often result from a Viability Analysis of the domain.

Step 4. Assess Domain Status

Action Evaluate the technical maturity of the domain in terms of Domain Objectives and plans for domain development and evolution.

Input

- Domain Plan.
- Domain Assumptions.

Result Domain Status.

Heuristics Domain Status and Viability Analysis must result in an endorsement of and commitment to a specific domain scope (set of assumptions). This endorsement may come directly from the intuitions of experienced personnel, or it may derive from a more extensive, quantitative analysis. Even in the latter case, however, all estimates and interpretations will rely on the best judgment of senior people. Many of the heuristics listed below are couched in qualitative terms to capture the essence of the decision being made, but you can always augment that decision by the suggested, optional quantitative analyses.

Determine Marketability

- Are systems of this sort marketable? Look at the Domain Synopsis. The Domain Synopsis impart an intuitive feel that it encompasses systems that the profiled customers will buy. If not, identify what is missing and add the missing elements to the Synopsis.
- Do the Commonality Assumptions correspond to the essential needs of the target customers? Are they really interested in systems that do everything implied by these assumptions, or may the scope be reduced? Conversely, are features they always require (possibly in variant forms) missing? Change the assumptions accordingly.
- Do the Variability Assumptions represent the real issues that determine whether a system addresses the individual needs of the target customers? Are there inconsequential variations (to the target customers) that you can constrain without losing business? Are there important differences in the needs of target customers that are not captured in the variabilities? Change the assumptions accordingly.

- Do expectations of target customers include not only those of the system end-user, but also the expectations of analysts and decision makers who influence awarding of contracts? In particular, the administrative expectations should include items such as allowable contract costs.

Determine Implementability and Risk

- A straightforward way to reduce the number of variations supported is first to decide what customers (or customer types) must be retained for viability of the business. Then determine must-have and nice-to-have variability alternatives for each of these customers. The aggregation of the must-have alternatives determines the minimal scope of your domain. Now you can consider all other proposed variability alternatives with regard to their incremental value added.

- Determine whether your organization has a good understanding of the problems such systems are intended to solve, compared with your competition, and whether your organization has authoritative expertise that will be committed to developing this domain.

- Determine whether your organization can build such systems. Consider whether it has built such systems in the past. Consider the success of such prior experience with particular regard to demonstrated mastery of the relevant software technology.

- If particularly difficult technical issues arise, determine which Domain Assumptions are affected. If you can reduce a high technical risk at the cost of removing or constraining an Assumption, consider how much domain value is lost by this reduction in scope and risk.

- Calculate a range for estimated system cost, potential market, anticipated income, and the like, representing your best- and worse-case expectations, since good estimates are typically hard to come by and are speculative in nature.

3.1.1 Risk Management

Risk There is a lack of critical expertise.

Implication The Domain Definition cannot be completed or there is unacceptably low confidence in the results.

Mitigation

- Commit time and resources to acquiring the expertise.

- Restrict Domain Assumptions sufficiently to reduce the need for expertise.

Risk The scope of the domain may be too narrow, thus precluding profitable variations.

Implication Opportunities for additional projects are lost.

Mitigation Review the Domain Definition with management, experienced engineers, and potential customers to identify additional variations.

Risk The scope of the domain may be too broad.

Implication Resources are misapplied to solving an unnecessarily general problem.

Mitigation Review the Domain Definition with management and experienced engineers to identify under-constrained Commonalities.

Risk Domain Assumptions are too precise or too vague.

Implication Flexibility is reduced unnecessarily, or key decisions are left to the discretion of domain engineers.

Mitigation Review the Domain Definition with management and experienced engineers to identify over- or under-constrained Domain Assumptions.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Domain Plan cannot be satisfied with available technical capabilities.

Source Domain Management.

Response Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete a Domain Definition that satisfies Domain Objectives as closely as possible.

Contingency The practices and procedures specified in the Domain Plan are either ineffective or inefficient.

Source Domain Management.

Response Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

4.2 Feedback From Product Consumers

Contingency The Domain Definition fails to provide the capabilities required by the Domain Plan.

Source Domain Management.

Response Evolve the Domain Definition to be consistent with the Domain Plan.

Contingency The Domain Definition is incomplete, ambiguous, or inconsistent.

Source Domain Specification Activity.

Response Revise the Domain Definition to correct the inadequacies.

This page intentionally left blank.

DE.3.2. DOMAIN SPECIFICATION ACTIVITY

1. GETTING STARTED

The Domain Specification Activity is an activity of Domain Analysis for creation of a Domain Specification. A Domain Specification is a precise characterization of the product family denoted by a domain and of a process for constructing members of that family. The product family is characterized from two perspectives: how problems are stated and how solutions are structured. Problems are expressed in the form of a requirements specification. Solutions are expressed in the form of a standardized design. Both forms are adaptable to anticipated variations in problems and solutions.

1.1 Objectives

The purpose of the Domain Specification Activity is:

- To create a precise specification of the problems and solutions supported by the domain.
- To define an application engineering process that is suited to the needs of building a product in the domain.

1.2 Required Information

The Domain Specification Activity requires the following information:

- Domain knowledge
- Domain Definition

1.3 Expertise Needs

The Domain Specification Activity requires the following expertise:

- Understanding of how systems in the domain are constructed, including the issues that application engineers must resolve to create a particular system.
- Understanding of concepts and structures that are convenient forms by which to communicate about the distinguishing features of systems in the domain.
- Expertise in the principles and use of appropriate requirements and design methods.

2. PRODUCT DESCRIPTION

Name Domain Specification.

Purpose The Domain Specification is a specification for a product family and an associated Application Engineering process for producing members of the family.

Content A Domain Specification consists of the following components:

- *Decision Model* (see Section DE.3.2.1), which identifies the Application Engineering requirements and engineering decisions that determine how members of the product family can vary.
- *Product Requirements* (see Section DE.3.2.2), which determines the behavior and operational characteristics of problems solved by the product family.
- *Process Requirements* (see Section DE.3.2.3), which determines how Application Engineering is performed and which work products are produced as a result.
- *Product Design* (see Section DE.3.2.4), which determines the structure and composition of solutions provided by the product family.

Verification Criteria

- Determine whether all aspects of the Domain Definition are accurately captured in the Domain Specification.
- Determine whether you can describe existing or envisioned systems in terms of the Domain Specification. Identify systems that exhibit behavior not indicated in the Domain Specification.

3. PROCESS DESCRIPTION

The Domain Specification Activity process, shown in Figure DE.3.2-1, consists of four activities:

- Decision Model Activity (see Section DE.3.2.1).
- Product Requirements Activity (see Section DE.3.2.2).
- Process Requirements Activity (see Section DE.3.2.3).
- Product Design Activity (see Section DE.3.2.4).

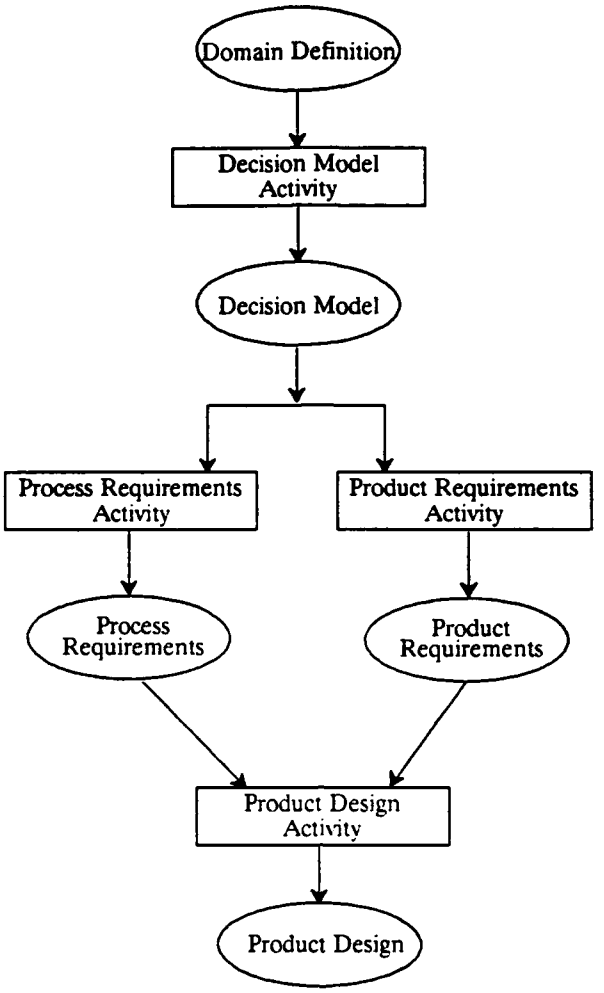


Figure DE.3.2-1. Domain Specification Activity Process

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Domain Definition is incomplete, ambiguous, or inconsistent.

Source Domain Definition Activity.

Response Describe the inadequacies in the Domain Definition. Proceed with Domain Specification, and document any assumptions made regarding the inadequate portions of the Domain Definition.

Contingency The Domain Plan cannot be satisfied with available technical capabilities.

Source Domain Management.

Response Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete a Domain Specification that satisfies the Domain Plan as closely as possible.

Contingency The practices and procedures specified in the Domain Plan are either ineffective or inefficient.

Source Domain Management.

Response Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

4.2 Feedback From Product Consumers

Contingency Suggestions are made for Domain Specification changes to exploit unforeseen opportunities. For example, a situation where substantial software is made available for use in the Domain Implementation that was not available when the Domain Specification was completed.

Source

- Product Implementation Activity.
- Process Support Development Activity.

Response

- Revise the Domain Specification.
- Refer to Domain Management for future planning.
- Reject the changes due to conflicts with the Domain Definition.

Contingency The Domain Specification fails to provide the capabilities required by the Domain Plan.

Source Domain Management.

Response Evolve the Domain Specification to be consistent with the Domain Plan.

Contingency The Domain Specification is incomplete, ambiguous, or inconsistent.

Source
- Product Implementation Activity.
- Process Support Development Activity.

Response Refine the Domain Specification to correct any inadequacies.

Contingency The standardized Application Engineering process is inefficient or leads to less-than-ideal results for a particular project.

Source Application Engineering.

Response
- Determine that the benefits of process standardization outweigh the interests of the particular project.
- Evolve the Application Engineering process to reflect this project's experience or to be more flexible under the particular conditions of concern.

This page intentionally left blank.

DE.3.2.1. DECISION MODEL ACTIVITY

1. GETTING STARTED

The Decision Model Activity is an activity of the Domain Specification Activity for producing a Decision Model. A Decision Model defines the set of requirements and engineering decisions that an application engineer must resolve in order to describe and construct a product. A Decision Model is an elaboration of a domain's Variability Assumptions and is the abstract form (i.e., concepts and structures) of an Application Modeling Notation (see Section DE.3.2.3). These decisions and the logical relationships among them determine the variety of products that make up the domain. For a product to be constructable, these decisions must be sufficient to distinguish the product from all other members of its product family. The decisions establish how work products of Application Engineering, including software and documentation, can vary in form and content.

1.1 Objectives

The Decision Model Activity defines a set of decisions that are adequate to distinguish among the members of a product family and to guide adaptation of associated Application Engineering work products.

1.2 Required Information

The Decision Model Activity requires the following information:

- Domain knowledge
- Domain Definition

1.3 Expertise Needs

The Decision Model Activity requires the following expertise:

- Conceptual modeling skills (similar to those needed to create a database conceptual schema) (see, for example, [Kent 1978; Borgida 1985]).
- Understanding of the issues that experienced engineers resolve when constructing systems in the domain.

2. PRODUCT DESCRIPTION

Name Decision Model.

Purpose

A Decision Model specifies the decisions that the Application Modeling Notation must allow an application engineer to make in describing an instance of the domain. These decisions determine the extent of variation that is possible among the systems of the domain. Equivalently these decisions determine the form and content of the work products that comprise each system.

Content

The Decision Model work product consists of three components:

- Decision specifications: Specifications of the set of decisions that is sufficient to distinguish among systems in the domain.
- *Decision groups*: A structuring of the decisions into logical groups.
- Decision constraints: A set of constraints on the resolution of interdependent decisions.

Verification Criteria

- Every decision must be an elaboration of one or more Variability Assumptions.
- The Decision Model must accommodate all Variability Assumptions.

Form and Structure

A decision specification takes one of two standard forms:

- List of questions
- Tabular format

An example of the question-list format is shown in the Decision Model work product in iteration 1 of the ATD/CWM case study (Volume 2, Section CS.1.1). Examples 1 and 2 show one form of tabular format. The Decision Model work product of iteration 4 of the ATD/CWM case study (Volume 2, Section CS.1.4) shows another.

Example 1. "Aircraft_Display_Symbol" Decision Group

Aircraft_Display_Symbol		
Decision	**Value Space**	**Description**
Host_Shape	enum of (circle, square, triangle)	Icon shape for the host_aircraft.
ID_Shape	composite of	Characterization of "identified" potential_threats.
partition	predicate	A test on the criteria that indicates whether a potential_threat is "identified."
shape	enum of (circle, square, triangle)	Icon shape for "identified" potential_threats.
UID_Shape	enum of (circle, square, triangle)	Icon shape for "unidentified" potential_threats.
. . .		

Table DE.3.2.1-1. Value Specifiers

Value Specifier	What the Application Engineer Can Specify
identifier	Sequence of A-Z, a-z, 0-9, or _.
string	Sequence of printable ASCII characters.
integer [(low .. high)]	Integer (restricted to the specified range).
real [(low .. high , resolution)]	Real (restricted to the specified range/resolution).
enum of (ident1, ident2, ..., identN)	One of the identifiers.
decision class	An instance of the decision class as defined in the Decision Model.
predicate	A boolean-valued expression that tests the value of data items defined in Product Requirements.
composite of	All of the component decisions.
alternative of	One of the component decisions.
list of 'value specifier'	A list of items characterized by "value specifier."
. . .	

If a set of related decisions is always resolved as a unit, you can define the set to be a composite decision. Composite decisions are shown in tabular form using a combination of the **composite of** indicator and indentation (see Example 1). If the application engineer can choose to resolve one (and only one) decision from a set of alternatives, you can define the set to be an alternative decision. Alternative decisions are shown in tabular form using a combination of the **alternative of** indicator and indentation.

You can also use a tabular format to specify constraints on decision making. Constraints may be either structural (see Example 3) or dependency (see Example 4). In both cases, a decision group (the Decision Group column) is specified as the focus of the constraint. A *structural constraint* is a constraint that limits the number of instances of a decision group in an Application Model. Valid entries include **exactly-one**, **one-or-more**, **zero-or-one**, **zero-or-more**, and **one-for-each** X, where X corresponds to other identified decision groups. A *dependency constraint* is a constraint that specifies how decisions made by an application engineer affect subsequent decisions. In Example 4 the column labeled Condition specifies a predicate (optionally referring to particular decisions of the decision group) that determines whether the constraint holds for a particular Application Model.

Example 3. Structural Constraints

Decision Group	Structural Constraint
Surveillance_Area	exactly-one
Collision_Warning_Situation	one-or-more
Collision_Warning_Situation_Response	one-for-each Collision_Warning_Situation
Aircraft_Status_Display	one-for-each Collision_Warning_Situation/ (identified, unidentified) pairwise combination
ATC_Format	zero-or-one
Aircraft_Display_Symbol	exactly-one
Host_Aircraft_Status_Display	one-for-each Collision_Warning_Situation
. . .	

Example 4. Dependency Constraints

Decision Group	Condition	Dependency Constraint
Collision_Warning_Situation_Response	True	At least one Collision_Warning_Situation_ Response must have the Corrective_Msg decision of True.
ATC_Format	True	A decision must be made for the ATC_Msg message format only when at least one Collision_Warning_Situation_ Response has the ATC_Msg decision of True.
. . .		

3. PROCESS DESCRIPTION

3.1 Procedure

These steps are followed for Decision Model Activity.

Step 1. Identify Decisions

Action Identify the decisions that an application engineer may make.

Input Variability Assumptions.

Result Decisions specification: A specification of a set of decisions sufficient to distinguish among systems in the domain.

Heuristics

- You can derive many decisions directly from Variability Assumptions. You must have (at least) one decision for each variation specified in the assumptions. The decision allows the application engineer to specify how the associated variation is to be resolved in a desired system. It is likely that you will derive multiple decisions from a single Variability Assumption; each decision is an elaboration of some aspect of the basic variability.

- Keep in mind that the relevant decisions are those concerning system generation time, rather than run-time, variation. If you followed a similar heuristic in identifying Domain Assumptions, such decisions should not be an issue here. Your focus now should be on how separate systems differ, rather than on ways in which systems vary their behavior at run-time. On the other hand, if systems differ in having variable run-time behavior, a valid decision may concern whether or how a particular system varies its behavior.

- If a Variability Assumption asserts that a certain characteristic of systems in the domain is variable without saying exactly how it varies, you must determine exactly how the characteristic can vary. Specify the precise type of information that will resolve a decision.

- Avoid routinely providing decisions that dictate arbitrary implementation limits (e.g., maximum number of users) unless those limits reflect a policy decision. Optimization of a system requires adequate flexibility.

Step 2. Structure Decisions

Action Organize decisions into logically-related and interconnected groups.

Input Decisions specification.

Result Decision groups: A partitioning of the decisions into a network of logical groups.

Heuristics

- Each decision group should represent a coherent and cohesive concept to domain experts. Such concepts usually have recognizable names. A concept may be independent of other concepts, or may be an aggregate concept that unifies other simpler concepts. In other words, a decision group may include both individual decisions and decision groups as elements.

- Structure the set of decisions based on the principle of separation of concerns (Dijkstra 1972), e.g., group decisions that correspond to features of a single, physically-distinct entity.

- Group together mutually-dependent decisions, i.e., those that are unlikely to change independently. Domain experts often rely on a single concept that ties dependent decisions together.

- Group together decisions that repeat. For example, if you need to describe multiple types of a particular device, the engineer may make similar decisions for each type. You can group these decisions to create a single concept as a focus for decisions.

- Group together decisions if they derive either from a corresponding single Variability Assumption or from separate assumptions that were grouped in the Domain Definition. A single assumption that motivates several decisions often represents a single concept, while assumption groupings often suggest how domain experts organize their thoughts about such systems.

- The principles of database schema normalization is a valid model for this step. As is the case with normalization, the goal here is to identify and organize a set of concepts without redundancy or inconsistency.

Step 3. Identify Constraints

Action Define structural and dependency constraints that limit how decisions are resolved.

Input Decision groups.

Result Decision Model.

Heuristics

- Define a structural constraint for each decision group; specify limits on when the group can appear in a valid Application Model.

- Define a dependency constraint whenever one decision narrows the resolution that the application engineer can provide for another decision.

3.2 Risk Management

Risk The Decision Model is inadequate for descriptions of intended systems.

Implication The domain will not provide effective support for planned projects.

Mitigation Try to describe one or more existing systems in terms of the Decision Model. Review these descriptions with experienced engineers to identify erroneous assumptions or unacceptable limitations.

Risk The decision space is too large or complex.

Implication Effort required to develop the Decision Model and subsequent adaptable work products will exceed a reasonable level.

Mitigation

- Focus on a set of well-understood decisions and make the assumption explicitly that the other decisions have fixed values (i.e., temporarily constrain them to be commonalities). Plan to relax these assumptions in subsequent iterations, or, in extreme cases, suggest that the Domain Definition Activity consider narrowing the domain scope.
- Reorganize the decision space to achieve a more effective separation of concerns.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Domain Definition is incomplete, ambiguous, or inconsistent.

Source Domain Definition Activity.

Response Describe the inadequacies in the Domain Definition and suggest appropriate refinements. Proceed with Decision Model, and document any assumptions made regarding the inadequate portions of the Domain Definition.

Contingency The Domain Plan cannot be satisfied with available technical capabilities.

Source Domain Management.

Response Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete a Decision Model that satisfies the Domain Plan as closely as possible.

Contingency The practices and procedures specified in the Domain Plan are either ineffective or inefficient.

Source Domain Management.

Response Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

4.2 Feedback From Product Consumers

Contingency The Decision Model fails to support all the variation described in the Domain Definition.

Source Domain Management.

Response Refine the Decision Model to be consistent with the Domain Definition.

Contingency The Decision Model is incomplete, ambiguous, or inconsistent.

Source
- Product Requirements Activity.
- Process Requirements Activity.
- Product Design Activity.
- Process Support Development Activity.

Response Refine the Decision Model to correct inadequacies.

Contingency The structure or content of the Decision Model conflicts with domain expert's conception of an Application Model.

Source Process Requirements Activity.

Response Refine the Decision Model to support an Application Modeling Notation acceptable to domain experts.

This page intentionally left blank.

DE.3.2.2. PRODUCT REQUIREMENTS ACTIVITY

1. GETTING STARTED

The Product Requirements Activity is an activity of the Domain Specification Activity for the creation of *Product Requirements*. A software-requirements specification describes a problem to be solved by application software. A problem is described in terms of the required behavior and operational environment of that software. Analogously, Product Requirements is a software-requirements specification that is adaptable to the decisions supported by the domain's Decision Model. It describes the set of problems solved by the members of a product family. By applying the decisions that characterize a particular product (i.e., its Application Model) to the Product Requirements, a standardized software-requirements specification is produced for that product. Product Requirements define the meaning of an Application Model as a description of a member of a product family.

1.1 Objectives

The Product Requirements Activity creates a software-requirements specification for a product family. The specification must be adaptable to decisions allowed by the Decision Model.

1.2 Required Information

The Product Requirements Activity requires the following information:

- Domain knowledge
- Domain Definition
- Decision Model

1.3 Expertise Needs

The Product Requirements Activity requires the following expertise:

- Understanding of the issues that application engineers must resolve in constructing systems in the domain.
- Understanding of concepts and structures that are appropriate for describing the behavior and operational environment of systems in the domain.
- Expertise in the principles and use of an appropriate software requirements specification method (e.g., informal, structured, semi-formal, or formal) (Heninger 1980).

2. PRODUCT DESCRIPTION

Name Product Requirements.

Purpose A Product Requirements specifies the software requirements of members of a product family. Given an Application Model that accurately characterizes a particular product, you can apply the decisions in that Model to the Product Requirements to derive a software-requirements specification for that product. The Product Requirements also ascribes meaning to Application Models created in accordance with the corresponding Decision Model. You can use it to understand (and explain to application engineers) the implications of decisions in an Application Model as they affect the behavior and operational environment of the product software.

Content A Product Requirements is a parameterized description of software for a product in the domain, including:

- *Implicit requirements*: The requirements that characterize any member of a product family as determined strictly from the Commonality Assumptions for the domain.
- *Derived requirements*: The requirements that characterize a particular product as implied by the decisions allowed by the Decision Model.

COMMENT: A black-box description for Product Requirements reduces the tendency to limit software design and implementation solutions prematurely. By parameterizing the description, it will apply equally well to all members of the product family. Figure DE.3.2.2-1 illustrates how an Application Model for a product is applied to a parameterized Product Requirements to yield a standardized software-requirements specification for that product.

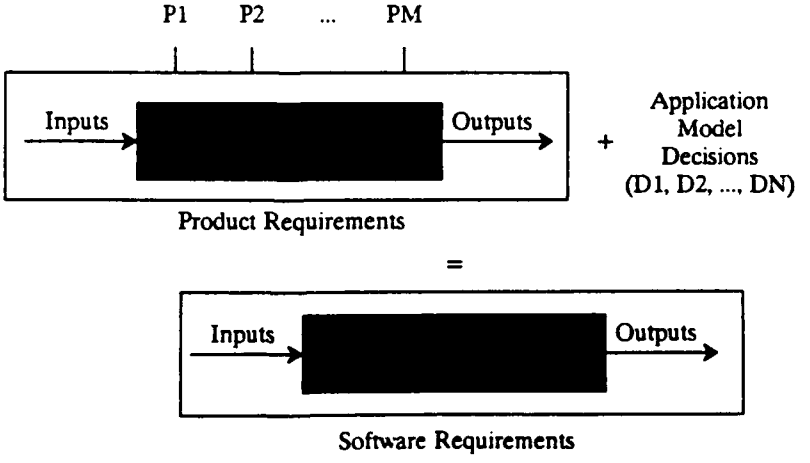


Figure DE.3.2.2-1. Instantiating Product Requirements

Form and Structure

Product Requirements may be expressed in any of three standard ways or forms:

- Structured, informal text.
- Assertions.
- A formal or semi-formal specification.

The assertions form of Product Requirements is a set of assertions that describe the (black-box) behavior of systems in the domain. Implicit requirements occur as simple assertions. Derived requirements occur as parameterized assertions whose parameters are decisions defined in the Decision Model. You can structure assertions into a hierarchy to allow separation of concerns.

Examples of semi-formal specification approaches are found in (Heninger 1980) and (Faulk et al. 1991). Appendix CS.1.A in Volume 2, presents an example developed for later iterations of the ATD/CWM case study.

For all forms, you should use parameterization to express the effect of decisions on Product Requirements. You can accomplish parameterization by using a metaprogramming notation (see Section OV.1.6) to indicate text substitution, conditional inclusion, and iteration over repetitive decisions. Example 1 illustrates a set of assertions that are parameterized using decisions from the Decision Model work product for iteration 4 of the ATD/CWM case study (Volume 2, Section CS.1.4).

Verification Criteria

- All implicit requirements must be an elaboration of one or more Commonality Assumptions.
- The Product Requirements must elaborate all Commonality Assumptions.
- If decisions that characterize a particular system are applied to the Product Requirements, the result should be a requirements specification that describes that system correctly.

Example 1. Parameterized Assertions

< foreach C in Collision_Warning_Situation >
When a potential threat enters collision warning situation **< C.cws_name >** relative to the host_aircraft, the following actions occur.

< if C.response.alarm then >
An audible alarm sounds in the host aircraft's cockpit at frequency 1,000 hertz for 0.5 seconds. The alarm only sounds when the potential threat migrates from a lower priority collision warning situation to one of higher priority.
< endif >

Example 1, continued

<if C.response.atc_msg then>
A message is sent to the nearest air traffic control center. The message contains a four-digit transponder code and the altitude of the host_aircraft.
<endif>

<if C.response.inter-air_msg then>
A message is sent to the potential threat.
<endif>

<if C.response.corrective_msg then>
A corrective action advisory message is displayed on the ATD located in the host aircraft's cockpit.
<endif>

<endfor>

3. PROCESS DESCRIPTION

3.1 Procedure

These steps are followed for the Product Requirements Activity.

Step 1. Structure Product Requirements

Action Create a hierarchical structure for describing systems that correspond to domain Commonality Assumptions based on the principle of separation of concerns.

Input Domain Assumptions.

Result Product Requirements Structure: A framework for describing systems in the domain.

Heuristics

- Decide upon a requirements method that best supports an abstract description of systems in the domain. The method determines the questions that an engineer must answer to characterize a system properly and the form in which to record the answers to create a requirements specification. A Product Requirements is nothing more than a parameterization of the form of a requirements specification that allows the expression of variations appropriate to the family of systems.

- Establish a structure (i.e., an outline) for describing systems that organizes assumptions into categories that correspond to major areas of concern to domain experts. Use the principle of separation of concerns as a guide. A reasonable organization discriminates among environment, external (observable) behavior, and constraints on internal behavior.

Step 2. Derive Common Product Requirements

Action Elaborate the Product Requirements structure to create a description that expresses all Commonality Assumptions precisely.

Input

- Domain Assumptions.
- Product Requirements Structure.

Result Common Product Requirements: A description of the requirements common to all systems in the domain. A description is partial in that it does not describe requirements that vary across the domain. These requirements define the implicit Product Requirements of systems within the domain.

Heuristics Common Product Requirements describe the requirements to be satisfied by any system in the domain. Product Requirements that are common to all systems in the domain properly characterize any system.

Step 3. Parameterize Product Requirements

Action Parameterize the Common Product Requirements to make it adaptable to supported decisions.

Input

- Decision Model.
- Common Product Requirements.

Result Adaptable Product Requirements: A parameterized specification of the requirements on systems in the domain. The description is complete in that it describes both common and varying requirements across the domain.

Heuristics

- To characterize the distinctive requirements of a particular system, tailor Product Requirements according to the decisions allowed by the Decision Model. These decisions, viewed as parameters of the description, allow controlled conditional and repetitive adaptation of the Product Requirements to derive the requirements for a system. The decisions characterizing a particular system are represented in an Application Model.
- Most decisions will have some effect on the Product Requirements of systems in the domain. Other decisions correspond either to Application Engineering documentation that would have no effect on how a system is constructed, or to design concerns which affect the design and implementation of a system, but not its requirements.

3.2 Risk Management

Risk The Product Requirements does not capture all Domain Assumptions accurately.

Implication A derived requirements specification will not be an accurate description of a system.

Mitigation Create an Application Model for one or more existing systems and derive their respective requirements specifications. Review with customers and experienced engineers to identify any inaccuracies.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Domain Definition is incomplete, ambiguous, or inconsistent.

Source Domain Definition Activity.

Response Describe the inadequacies in the Domain Definition. Proceed with Product Requirements, and document any assumptions made regarding the inadequate portions of the Domain Definition.

Contingency The Domain Plan cannot be satisfied with available technical capabilities.

Source Domain Management.

Response Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete Product Requirements that satisfy the Domain Plan as closely as possible.

Contingency The practices and procedures specified in the Domain Plan are either ineffective or inefficient.

Source Domain Management.

Response Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures that will make them more effective.

Contingency The Decision Model is incomplete, ambiguous, or inconsistent.

Source Decision Model Activity.

Response Describe the inadequacies in the Decision Model. Proceed with Product Requirements and document any assumptions made regarding the inadequate portions of the Decision Model.

4.2 Feedback From Product Consumers

Contingency The Product Requirements fail to describe a family of systems that is consistent with the Domain Definition.

Source Domain Management.

Response Modify the Product Requirements to be consistent with the Domain Definition.

Contingency The Product Requirements are incomplete, ambiguous, or inconsistent.

Source

- Product Design Activity.
- Product Implementation Activity.

Response Refine the Product Requirements to correct inadequacies.

This page intentionally left blank.

DE.3.2.3. PROCESS REQUIREMENTS ACTIVITY

1. GETTING STARTED

The Process Requirements Activity is an activity of the Domain Specification Activity for creating Process Requirements. Process Requirements is a specification of a standardized Application Engineering process for a domain and an associated Application Modeling Notation. A process specification tailors and standardizes the activities, methods, and procedures by which Application Engineering is practiced within a domain. This process replaces conventional, domain-independent approaches to software development.

As a part of a standardized Application Engineering process, application engineers create an Application Model that describes a required system. An Application Modeling Notation defines the form and mechanisms that application engineers can use to describe and evaluate an Application Model for products in the domain. This notation must be usable by engineers knowledgeable in domain concepts and must produce an Application Model that is an accurate model of the resulting product. The information content expressible with an Application Modeling Notation for a domain must be equivalent to the information content defined by the Decision Model for that domain. The organization and form of the notation, however, are tailored to the needs of application engineers.

1.1 Objectives

The purpose of the Process Requirements Activity is to define a standardized process for Application Engineering within a domain and an accompanying Application Modeling Notation for creating a precise description of a required product. The process should be tailored to the needs of the organization so that established productivity (process efficiency and product quality) goals are most attainable.

1.2 Required Information

The Process Requirements Activity requires the following information:

- Domain knowledge
- Domain Definition
- Decision Model

1.3 Expertise Needs

The Process Requirements Activity requires the following expertise:

- Understanding of the conventional life-cycle process of systems and the role of customers and standards in that process.

- Understanding of how application engineers resolve issues in constructing systems in the domain.

- Understanding of concepts and structures that are convenient forms by which domain experts communicate concerning the distinguishing features of systems in the domain.

2. PRODUCT DESCRIPTION

Name Process Requirements.

Purpose Process Requirements define a standard process that Application Engineers follow to develop and evolve systems in the domain. The process described in the Process Requirements may be manual or may incorporate varying levels of automation.

Content The Process Requirements work product consists of:

- Process Specification: A definition of the work products, activities, and process of Application Engineering.

- Application Modeling Notation Specification: A description of the form of the Application Modeling Notation. The description of the notation consists of:

    a. Presentations: The form of each set of logically-related decisions from the Decision Model that Application Engineers tend to treat as a unit.

    b. Structure: The structure of the Notation, which organizes Presentations into a decision process based on dependencies and constraints among decisions.

Form and Structure The Process Specification defines the products, activities, and process of Application Engineering. Each Application Engineering work product has a specified content and form. The form of a product may be defined explicitly or by reference to customer or industry standards. Each activity of Application Engineering is described by its purpose, the work products to be created, a procedure that defines and organizes the steps of the activity, and interactions with other activities. The Application Engineering process is described as a set of Activities (e.g., Specify Model. Assess Model) that are to be performed in a specified (partial) order. Each activity consists of a number of Steps (e.g., Specify Platform and Start Simulation). The description of the activity includes a specification of the order in which steps may be performed. Each step is specified in terms of a presentation that describes the particular information the Application Engineer sees during that step and a set of actions that he can perform.

The Application Modeling Notation can be specified as a set of paper or automated forms that allow Application Engineers to make requirements or engineering decisions about the needed product. By completing these forms, Application Engineers construct an Application Model for a product. In addition, the Notation organizes the forms into a network that defines a sequence in which Application Engineers can address any particular decision.

Process Requirements for the ATD/CWM domain are shown in Volume 2, Section CS.1.4.

The description of an activity in an Application Modeling Notation is written in terms of a standard presentation paradigm. A presentation paradigm defines a generic form in which information may be presented interactively and is conceived as a notational aid for describing the Application Modeling Notation. Presentation paradigms for simple decisions are textual, graphical, or iconic. Presentation paradigms for grouped decisions organize simpler presentations into aggregate (possibly hybrid) forms (e.g., textual, graphical, tabular). The presentation paradigms used in the ATD/CWM Process Requirements are defined in the Process Requirements work product of Volume 2, Section CS.1.4.

Verification Criteria

- All decisions specified in the Decision Model should be accessible through the Application Modeling Notation.
- The Application Modeling Notation should support only those decisions that can be expressed in some equivalent way in the Decision Model.
- The Process Requirements must enforce all of the constraints specified in the Decision Model.

3. PROCESS DESCRIPTION

3.1 Procedure

These steps are followed for the Process Requirements Activity.

Step 1. Develop Application Modeling Notation

Action Organize decisions into a set of presentations in terms of a set of standard presentation paradigms.

Input Decision Model.

Result

- Application Modeling Notation Specification: A specification of the way decisions are organized into units and structured to form an Application Model.
- Paradigm Specifications: Specifications for standard forms used to present information that is meaningful to application engineers in a particular domain.

Heuristics

- Identify a set of presentation paradigms; consider the ways domain experts generally represent various problem facets in a manner consistent with the ways decisions are grouped in the Decision Model. Identify a set of paradigms that are adequate to represent decisions in all the facets of a problem description.
- First, identify presentations by describing each decision group in the Decision Model as a presentation, then describe the presentation in terms of allowed decisions and any auxiliary (e.g., labeling, layout) information required by the appropriate presentation paradigm.
- Create the structure of the Application Modeling Notation by considering how presentations interrelate (derivable from how decision groups in the Decision Model interrelate). Some presentations may be meaningful only if accessed in the context of, or in combination with, other related presentations. These relationships determine a reasonable structure for the Notation. The ideal structure for an Application Model is one that domain experts would consider natural and appropriate as a model of a system. Refine the structure by consulting with domain experts as to how decision making is best organized.
- Identify any analyses that the Application Engineer should be able to perform on individual presentations and on the Application Model as a whole. To be most effective, these analyses should provide insights into the functional and operational characteristics of the described system in terms of alternative resolutions *of decisions*, rather than in terms of the details of generated work products.

Step 2. Design a Process

Action Specify a process for the creation and evaluation of an Application Model and the generation of work products from the Model. Define the process by organizing the Presentations defined in the Application Modeling Notation into Activities and specifying ordering constraints and prerequisites that structure the process. Define the points at which analyses may be performed and work products may be produced.

Input Domain Definition.

Result Process Specification: A specification for an Application Engineering process.

Heuristics

- Identify the *deliverables* that the Application Engineering process must produce. This set of products is determined by the needs of customers. The form and content of each product should be defined in keeping with corporate, customer, or industry standards, as appropriate. If an adequate standard is not available for any product, create a standard for your organization.
- Identify additional (i.e., intermediate) work products that should result from the Application Engineering process. These work products support the need for quantitative and qualitative analyses of deliverable work products and the Application Engineering process.

- Define the activities to be performed during the process in terms of necessary inputs, expected results, and procedures for attaining those results. Identify risks in attaining acceptable results and interdependencies with other activities. Define the checks to be performed during the process and specify when they are to be performed. Specify what is to be done when checks fail.

- The process should be described to a level of detail that allows you to formalize standard practice and the engineer's past experience. For example, Project Management, Delivery and Operation Support, and configuration management practices may be based on conventional experience with those tasks.

- Creating an Application Model and using that Model to create work products are essential elements of any Application Engineering process. As a starting point, consider the prototypical, iterative Application Engineering process described in Section AE.1 (consisting of Project Management, Application Modeling, Application Production, and Delivery and Operation Support activities), refined appropriately to the needs of your projects.

3.2 Risk Management

Risk The Application Engineering process will not meet all of the needs of a project.

Implication Projects will have to modify the process in an *ad hoc* fashion and work around incorrect assumptions of the Process Requirements.

Mitigation Review the process with experienced project managers to ensure that it encompasses all activities required of a project, those products that customers require, and those products that benefit a project. Variations in project needs should be anticipated and supported.

Risk The Application Modeling Notation will not be usable by Application Engineers.

Implication Application Engineers will have difficulty creating valid Application Models.

Mitigation Review the Notation with experienced engineers to ensure that it is understandable and that it uses terminology and notations familiar to Application Engineers.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Domain Definition is incomplete, ambiguous, or inconsistent.

Source Domain Definition Activity.

Response Describe the inadequacies in the Domain Definition. Proceed with Process Requirements, and document any assumptions made regarding the inadequate portions of the Domain Definition.

Contingency The Domain Plan cannot be satisfied with available technical capabilities.

Source Domain Management.

Response Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete the Process Requirements to satisfy the Domain Plan as closely as possible.

Contingency The practices and procedures specified in the Domain Plan are either ineffective or inefficient.

Source Domain Management.

Response Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

Contingency The Decision Model is incomplete, ambiguous, or inconsistent.

Source Decision Model Activity.

Response Describe the inadequacies in the Decision Model. Proceed with Process Requirements, and document any assumptions made regarding the inadequate portions of the Domain Definition.

Contingency The structure or content of the Decision Model conflicts with domain expert's conception of an Application Model.

Source Decision Model Activity.

Response Describe the elements of an acceptable Application Modeling Notation not supported by the Decision Model.

4.2 Feedback From Product Consumers

Contingency The Process Requirements violate constraints on the Application Engineering process established in the Domain Definition.

Source Domain Management.

Response Evolve the Process Requirements to be consistent with the Domain Definition.

Contingency The Process Requirements are incomplete, ambiguous, or inconsistent.

Source Process Support Development Activity.

Response Refine the Process Requirements to correct inadequacies.

Contingency The standardized Application Engineering process is inefficient or leads to less-than-ideal results for a particular project.

Source Application Engineering.

Response

- Determine that the benefits of process standardization outweigh the interests of the particular project.
- Evolve the Process Requirements to reflect the generally-applicable insight of this project's experience or to be adapted to the particular conditions of concern.

This page intentionally left blank.

DE.3.2.4. PRODUCT DESIGN ACTIVITY

1. GETTING STARTED

The Product Design Activity is an activity of the Domain Specification Activity for creating a Product Design. A product consists of a set of work products. A design specifies a product as a solution to a problem by describing the structure and features of each of the work products (e.g., software or documentation). A Product Design is a design that is adaptable to the decisions supported by the domain's Decision Model. By applying the decisions that characterize a particular product (i.e., its Application Model) to the Product Design, a standardized design is created for that product. Process Requirements identify the work products that characterize a product and that must be produced by the Application Engineering process.

1.1 Objectives

The Product Design Activity creates a design for a product family, including a design for each of the work products required by Process Requirements for a product. This design must satisfy Product Requirements and be adaptable to the decisions allowed by the Decision Model.

1.2 Required Information

The Product Design Activity requires the following information:

- Domain knowledge
- Decision Model
- Product Requirements
- Process Requirements
- Existing components

1.3 Expertise Needs

The Product Design Activity requires the following expertise:

- Expertise in the principles and use of an appropriate design method.
- Understanding of how systems in the domain are designed, including an appreciation of typical engineering trade-offs to be resolved.

- Understanding of the concepts and practice of abstraction-based reuse (Parnas 1976; Campbell 1989; Software Productivity Consortium 1991b).

2. PRODUCT DESCRIPTION

Name Product Design.

Purpose A Product Design specifies the design of members of a product family. Given an Application Model for a product, you can apply the decisions in that model to the Product Design to derive a design for the work products that comprise the particular product.

Content The Product Design consists of three parts:

- *Product Architecture* (see Section DE.3.2.4.1): A specification of the structure of each application work product that can be produced for the family.
- *Component Design* (see Section DE.3.2.4.2): A specification of the design of each of a set of *Adaptable Components* that can be adapted and composed to create application work products for the family.
- *Generation Design* (see Section DE.3.2.4.3): A specification of how an Application Model is transformed into required application work products.

Verification Criteria

- All aspects of Product Requirements are traceable into the Product Design. All variations in Product Requirements have equivalent variations in the Product Design.
- The Product Design can be used to produce all the products specified in the Process Requirements.
- The combination of Product Architecture and the associated set of Component Designs satisfies the verification criteria appropriate to the specific design method used in creating them.

3. PROCESS DESCRIPTION

The Product Design Activity consists of three subactivities:

- Product Architecture Activity (see Section DE.3.2.4.1): Design the structure of each of the work products produced by the Application Production Activity.
- Component Design Activity (see Section DE.3.2.4.2): Design the components from which application work products are produced.
- Generation Design Activity (see Section DE.3.2.4.3): Define how decisions in an Application Model determine the form and content of each application work product.

The information flow among these activities is shown in Figure DE.3.2.4-1.

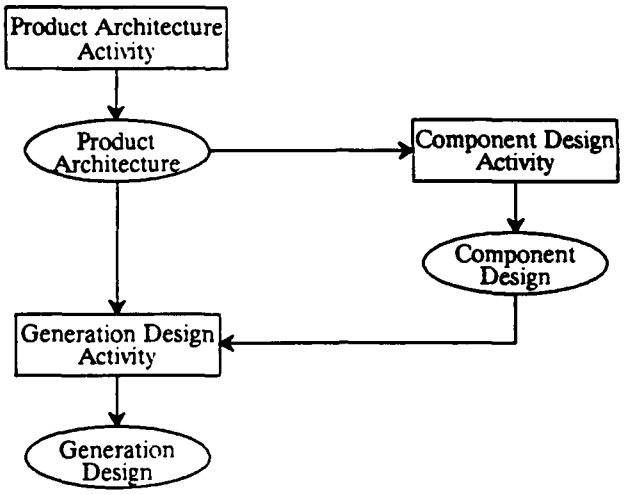


Figure DE.3.2.4-1. Product Design Process

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Decision Model is incomplete, ambiguous, or inconsistent.

Source Decision Model Activity.

Response Describe the inadequacies in the Decision Model. Proceed with Product Design, and document any assumptions made regarding the inadequate portions of the Decision Model.

Contingency The Product Requirements are incomplete, ambiguous, or inconsistent.

Source Product Requirements Activity.

Response Describe the inadequacies in the Product Requirements. Proceed with Product Design, and document any assumptions made regarding the inadequate portions of the Product Requirements.

Contingency The Domain Plan cannot be satisfied with available technical capabilities.

Source Domain Management.

Response Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete a Product Design that satisfies the Domain Plan as closely as possible.

Contingency The practices and procedures specified in the Domain Plan are either ineffective or inefficient.

Source Domain Management.

Response Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

4.2 Feedback from Product Consumers

Contingency Suggestions are made for Product Design changes to exploit unforeseen opportunities, e.g., a situation where substantial software is made available for use in the Domain Implementation that was not available when the Domain Specification was completed.

Source

- Product Implementation Activity.
- Process Support Development Activity.

Response

- Revise the Product Design.
- Refer to Domain Management for future planning.
- Reject the changes due to conflicts with the Domain Definition.

Contingency The Product Design does not satisfy the Product Requirements.

Source Verification Activity.

Response Modify the Product Design to be consistent with the Product Requirements.

Contingency The Product Design is incomplete, ambiguous, or inconsistent.

Source Product Implementation Activity.

Response Refine the Product Design to correct inadequacies.

DE.3.2.4.1. PRODUCT ARCHITECTURE ACTIVITY

1. GETTING STARTED

The Product Architecture Activity is an activity of the Product Design Activity for creating a Product Architecture. An *architecture* is one or more design structures that characterize the organization of a work product from different perspectives. A Product Architecture is a specification of a standardized architecture for each of the work products for the members of a product family.

For each required application work product, one design structure must identify a set of Adaptable Components, instances of which may be composed to create instances of that work product. As such, the Product Architecture specifies an organizational structure for each work product. Depending on which design method you follow, other structures are also created which provide other views of the behavior or interrelationships of components. In all cases, the structures are in an adaptable form so that Application Engineering can use them to produce any member of the indicated product family.

1.1 Objectives

The goal of the Product Architecture Activity is to create an adaptable architecture for products that can be produced in Application Engineering.

1.2 Required Information

The Product Design Activity requires the following information:

- Domain knowledge
- Product Requirements
- Process Requirements
- Existing components

1.3 Expertise Needs

The Product Architecture Activity requires the following expertise:

- Expertise in the principles and use of an appropriate software design method (e.g., ADARTS [Software Productivity Consortium 1991a]).
- Understanding of how systems in the domain are designed and documented, following the chosen design method.

- Understanding of typical engineering trade-offs that must be balanced when designing systems in the domain.

- Understanding of the concepts and practice of *metaprogramming* (Software Productivity Consortium 1991b).

2. PRODUCT DESCRIPTION

Name Product Architecture.

Purpose The Product Architecture describes the structure of the application work products that can be produced for systems in the family.

Content The Product Architecture work product consists of one or more design structures for each application work product required in the Process Requirements. Each of these structures consists of:

- A set of design elements.

- A relation that associates elements.

For example, the elements of a document are sections, and the relation is subsections; the elements of a *test scenario* are test cases, and the relation is sequencing. The structures developed for a particular domain depend on the particular application work products and the design method used to produce them. While the nature of a structure is captured by its elements and relation, a structure may contain additional information.

Although the Product Architecture for a particular product may contain multiple design structures, there must be one structure that describes the decomposition of the product into work assignments (e.g., *modules*, sections). The elements of this structure correspond to components that are to be implemented by Adaptable Components.

The only difference between the design structures specified in the Product Architecture and those specified in a conventional design is that the Product Architecture is parameterized and adaptable, so that it describes the family of systems in the domain.

Form and Structure The form for each structure of a Product Architecture is a textual or graphic network of elements and relations, augmented with metaprogramming notation (Software Productivity Consortium 1991b). Metaprogramming notation is used to parameterize the structure for adaptation to variations.

Form and Structure The ATD/CWM case study (Volume 2, Section CS.1) applies the process and class structuring guidance of the ADARTS method (Software Productivity Consortium 1991c) to accomplish this activity. These ADARTS activities produce three structures: Information Hiding, Process, and Dependency. Each structure must be represented in an adaptable form that allows tailoring to the decisions in an Application Model that specifies a particular system. Examples of adaptable forms of each of these structures for the ATD/CWM domain are in the Product Design work product of Volume 2, Section CS.1.4. Adaptability is represented using the metaprogramming notation described in Section OV.1.5.

Verification Criteria

- Each Product Architecture structure satisfies the verification criteria established by the specific design method used in its creation.
- The Product Architecture defines all structures for the software and other work products required by the Process Requirements.

3. PROCESS DESCRIPTION

3.1 Procedure

The following procedure is followed for each work product required by the Process Requirements:

Step 1. Identify Work Product Components

Action Develop a structure that describes the organization of the work product as a set of components.

Input

- Process Requirements.
- Product Requirements.

Result Product Architecture: Organizational Structure.

Heuristics

- The Process Requirements define the required standards for the development of each work product. For software, Process Requirements define which design method is to be followed, which in turn determines the design structures required. Using ADARTS, as noted above, there are three structures: Information Hiding, Process, and Dependency. For documentation, an annotated outline, in which each section and subsection is listed as a component, is normally sufficient as the design for the work product. For test and delivery support, a hierarchical structure is needed that characterizes each test case or delivery function as a component.

- Each work product is broken down, or decomposed, into a set of components so that the work may be performed independently by multiple component implementors. Using a chosen design method, a structure must be created that determines this decomposition. Using ADARTS, the Information Hiding Structure determines a suitable decomposition into components.

- Each structure is parameterized by a set of decisions that characterize how it is adapted to represent a particular system. A parameterized structure defines a family of structures. The set of structure families must be parameterized consistently, both with each other and with the Product Requirements. The content of each structure must be consistent with the Product Requirements.

- Analyze existing systems: look for recurring patterns in their design structures. Recurring patterns suggest portions of a structure that do not vary.

- Use the Product Requirements to determine what characteristics are common to all systems in the domain. These characteristics should correspond to common structures identified in the analysis of existing systems.

- Each structure must be adaptable to requirements and design variations among systems. Parameterize the structure to characterize required variations. Use a metaprogramming notation to record the effects of variations. For example, a conditional statement referencing a variation indicates that a portion of a structure is included only for some systems.

Step 2. Develop Alternate Structures

Action Create any other structures required to define the work product fully.

Input

- Process Requirements.
- Product Requirements.

Result Product Architecture: Alternate Structures.

Heuristics

- The chosen design method for software, identifies other required design structures. Using ADARTS, these design structures are the Process Structure and the Dependency Structure. Alternate structures are not usually required for documents.

- Alternate structures impose constraints on the implementation of each component of the organizational structure. The design method characterizes these constraints.

- All structures must support the same variations. The organizational structure determines how these variations affect each component. Component variations must account properly for variations in relevant parts of the alternate structures.

3.2 Risk Management

Risk The Product Architecture will not support all features or variations in Product Requirements.

Implication The Product Architecture is not a correct solution to the Product Requirements.

Mitigation Review the Product Architecture with developers of the Product Requirements and experienced designers. Establish traceability of all required features to elements of the architecture. Evaluate whether variations that characterize different systems lead to proper architectural variations.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Decision Model is incomplete, ambiguous, or inconsistent.

Source Decision Model Activity.

Response Describe the inadequacies in the Decision Model. Proceed with Product Architecture, and document any assumptions made regarding the inadequate portions of the Decision Model.

Contingency The Product Requirements are incomplete, ambiguous, or inconsistent.

Source Product Requirements Activity.

Response Describe the inadequacies in the Product Requirements. Proceed with Product Architecture, and document any assumptions made regarding the inadequate portions of the Product Requirements.

Contingency The Domain Plan cannot be satisfied with available technical capabilities.

Source Domain Management.

Response Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete a Product Architecture that satisfies the Domain Plan as closely as possible.

Contingency The practices and procedures specified in the Domain Plan are either ineffective or inefficient.

Source Domain Management.

Response Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

4.2 Feedback from Product Consumers

Contingency Suggestions are made for Product Architecture changes to exploit unforeseen opportunities. For example, a situation where substantial software is made available for use in the Domain Implementation that was not available when the Domain Specification was completed.

Source
- Product Implementation Activity.
- Process Support Development Activity.

Response
- Revise the Product Architecture.
- Refer to Domain Management for future planning.
- Reject the changes due to conflicts with the Domain Definition.

Contingency The Product Architecture does not satisfy the Product Requirements.

Source Verification Activity.

Response Modify the Product Architecture to be consistent with the Product Requirements.

Contingency The Product Architecture is incomplete, ambiguous, or inconsistent.

Source
- Product Implementation Activity.
- Generation Design Activity.
- Component Design Activity.

Response Refine the Product Architecture to correct inadequacies.

DE.3.2.4.2. COMPONENT DESIGN ACTIVITY

1. GETTING STARTED

The Component Design Activity is an activity of the Product Design Activity for creating a Component Design. The Product Architecture identifies a set of Adaptable Components that are required to implement the work products of a product family. A Component Design is a design specification for one of these Adaptable Components. A set of Component Designs defines a library of Adaptable Components that may be adapted and composed to implement the work products of the product family. Each component must be designed to satisfy relevant aspects of the Product Requirements and all design structures of the Product Architecture.

1.1 Objectives

The goal of the Component Design Activity is to produce a design for an Adaptable Component in accordance with its role in the Product Architecture; the design must satisfy any allocated Product Requirements.

1.2 Required Information

The Component Design Activity requires the following information:

- Domain knowledge
- Product Requirements
- Product Architecture

1.3 Expertise Needs

The Component Design Activity requires the following expertise:

- Understanding of how components of systems in the domain are designed.
- Expertise in the principles and use of an appropriate design method.
- Understanding of the concepts and practice of abstraction-based reuse (Campbell 1989; Software Productivity Consortium 1991b).

2. PRODUCT DESCRIPTION

Name Component Design.

Purpose A Component Design is a specification for an Adaptable Component that can be used to construct a system or an associated application work product.

Content Each Component Design represents a family of components. A Component Design consists of two parts:

- Adaptation specification: The adaptation specification for an Adaptable Component describes the ways that the component can be tailored as supported by a set of parameters. Each parameter is named and typed to indicate the range of variations supported. Constraints identify invalid combinations of variations.

- Interface specification: The interface specification describes the desired characteristics of the implementation of the component. The exact content of the interface specification is particular to the component type and the design method used. In order to describe the entire family, the interface specification is parameterized with respect to the variations in the adaptation specification.

Form and Structure The adaptation and interface specifications each include textual and tabular information. The form of an adaptation specification is the same for all types of components and includes the following information:

- Name: Name of the Adaptable Component.

- *Instantiation Parameters: Adaptation parameters for the component,* including the name, type, and description of each parameter.

- Instantiation Constraints: Constraints on the instantiation of the Adaptable Component (e.g., constraints on the legal combination of parameter values).

The interface part of Adaptable Components is different for software and documentation. The content of the software interface is design-method specific. The following types of information are examples: definitions of interface programs (names, parameters, parameter types, returned values), definitions of exported types, descriptions of the effects of interface programs, assumptions about the environment in which the software is to be used.

The interface for a documentation component does not require the same type of detailed information; it consists primarily of a brief statement of the content of the component.

See the Adaptable Components work product in Volume 2, Section CS.1.4, for examples of software and document Adaptable Components developed in the ATD/CWM case study.

Verification Criteria

- The Component Design satisfies all structures of the Product Architecture.
- The Component Design satisfies the verification criteria established by the specific design method used for its creation.
- Each parameter in the adaptation specification of a Component Design either is derivable from the Decision Model or has a common value for all products in the domain.

3. PROCESS DESCRIPTION

For each work product defined in the Product Architecture, an organizational design structure identifies a set of Adaptable Components. Use the following procedure to implement each of those components.

3.1 Procedure

These steps are followed for the Component Design Activity.

Step 1. Define Component Adaptation Specification

Action Identify the variations that parameterize the Adaptable Component, and record constraints on legal combinations.

Input

- Product Architecture.
- Product Requirements.

Result Adaptation specification.

Heuristics

- Identify components in the Product Architecture that have the same form in all instances of the domain. These components have no associated variations (i.e., they are not adaptable).
- The adaptation specification of a component may be determined by identifying which roles the component must serve in the design, determining which adaptations are required so that it can fulfill these roles, and devising a set of parameters that can be used to indicate desired adaptations.
- The determination of necessary component adaptations may be approached from two directions. The first is to analyze existing instances of the component in the domain to see how they vary. The second is to analyze the Product Requirements to see how the component must vary to satisfy relevant requirements. In practice, you will use both of these approaches.

- Note that the decisions that parameterize components must derive from, but need not be, the particular decisions that are identified in the Decision Model. In general, there is a many-to-many relationship between Decision Model decisions and Component Design parameters. It is a good practice to specify parameters so that they correspond to the designer's intuition as to what design decisions affect the design of a component.

Step 2. Specify Component Interface

Action Specify the requisite properties for the implementation of each component.

Input

- Product Architecture.
- Adaptation Specification.

Result Interface specification.

Heuristics The properties that you must specify depend on the type of component and the design method used. Parameterize each component interface with the decisions from the component's adaptation specification so that it describes all instances of the component.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Decision Model is incomplete, ambiguous, or inconsistent.

Source Decision Model Activity.

Response Describe the inadequacies in the Decision Model. Proceed with Component Design, and document any assumptions made regarding the inadequate portions of the Decision Model.

Contingency The Product Requirements are incomplete, ambiguous, or inconsistent.

Source Product Requirements Activity.

Response Describe the inadequacies in the Product Requirements. Proceed with Component Design, and document any assumptions made regarding the inadequate portions of the Product Requirements.

Contingency The Domain Plan cannot be satisfied with available technical capabilities.

Source Domain Management.

Response Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete a Component Design that satisfies the Domain Plan as closely as possible.

Contingency The practices and procedures specified in the Domain Plan are either ineffective or inefficient.

Source Domain Management.

Response Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

Contingency The Product Architecture is incomplete, ambiguous, or inconsistent.

Source Product Architecture Activity.

Response Describe the inadequacies in the Product Architecture. Proceed with Component Design, and document any assumptions made regarding the inadequate portions of the Product Architecture.

4.2 Feedback from Product Consumers

Contingency Suggestions are made for Component Design changes to exploit unforeseen opportunities. For example, a situation where substantial software is made available for use in the Domain Implementation that was not available when the Component Design was completed.

Source

- Product Implementation Activity.
- Process Support Development Activity.

Response

- Revise the Component Design.
- Refer to Domain Management for future planning.
- Reject the changes due to conflicts with the Domain Definition.

Contingency The Component Design does not satisfy the Product Requirements.

Source Verification Activity.

Response Modify the Component Design to be consistent with the Product Requirements.

Contingency The Component Design is incomplete, ambiguous, or inconsistent.

Source

- Component Implementation Activity.
- Generation Design Activity.

Response Refine the Component Design to correct inadequacies.

DE.3.2.4.3. GENERATION DESIGN ACTIVITY

1. GETTING STARTED

The Generation Design Activity is an activity of the Product Design Activity for creating a Generation Design. A Generation Design is a specification for the production procedures that an application engineer uses to produce the work products for a member of a product family. A Generation Design defines a transformation (or mapping) from an Application Model that describes a product to the equivalent application work products. For each work product, a Generation Design specifies how to select and tailor Adaptable Components according to decisions in an Application Model and to compose them according to the organizational structure of that work product in the Product Architecture.

1.1 Objectives

The goal of the Generation Design Activity is to produce a specification for a production procedure that can be used to produce work products for a member of a product family. The specification establishes a correspondence between an Application Model and equivalent work products that implement the intent of the model correctly.

1.2 Required Information

The Product Design Activity requires the following information:

- Domain knowledge
- Decision Model
- Product Architecture
- Component Designs

1.3 Expertise Needs

The Generation Design Activity requires the following expertise:

- Understanding of how systems in the domain are designed.
- Expertise in the principles and use of an appropriate design method.
- Understanding of the concepts and practice of abstraction-based reuse (Campbell 1989; Software Productivity Consortium 1991b).

2. PRODUCT DESCRIPTION

Name Generation Design.

Purpose A Generation Design is a specification for a production procedure for creating application work products. In domain implementation, you will either implement the procedure as an automated product generator or document it as a manual procedure to be carried out by application engineers.

Content For each work product specified in the Process Requirements, a Generation Design relates the decisions from the Decision Model to the elements of its organizational structure defined in the Product Architecture. This accomplishes the following:

- Architecture mapping: A mapping of the organizational structure of a work product in the Product Architecture that is based on the decisions in the Decision Model (i.e., from Application Models). This mapping is used to determine the structure of a work product from an Application Model that describes a member of the product family.
- Component mapping: A mapping of each element of the organizational structure to an Adaptable Component that implements that element. This mapping is used to determine how each component of a work product is to be produced.
- Decision mapping: For each work product component, a mapping from the decisions in the Decision Model to the variations in the adaptability specification of each Adaptable Component needed to produce that component. This mapping is used to determine how each Adaptable Component is tailored to implement a work product.

Form and Structure A Generation Design is a list of Product Architecture organizational structure element/Adaptable Component pairs (i.e., the architectural mapping). Each pair may be included conditionally and/or iteratively (relative to decisions in the Decision Model). A conditional term indicates when the element is included in an instance work product. An iteration term indicates that several versions of an element may be included in an instance work product. The association of a Product Architecture component with an Adaptable Component indicates that the Adaptable Component is to be used to implement the Product Architecture Component (i.e., the Component Mapping). Association of decisions from the Decision Model with parameters of each Adaptable Component indicates how each component is to be tailored (i.e., the Decision Mapping).

Example 1 shows an element from one of these lists. The conditional term on the first line determines when the Audible Alarm Device Component from the Product Architecture is included as an element of ATD/CWM Application Software. The instantiation of the Audible_Alarm Adaptable Component indicates that this component is to be used to implement the Audible_Alarm component of the Application Software and that the Audible_Alarm Adaptable Component should be adapted with the indicated Decision Model decisions associated with each potential collision-warning situation and its response.

Example 1. ATD/CWM Generation Design Example

```
if there exists C in CWS such that C.Response.Alarm then

5. Audible Alarm

    Audible_Alarm (ring  =  (
                    foreach C in CWS
                        if C.Response.Alarm then
                            (   cws_name: <C.CWS_Name>,
                                frequency: <C.Response.Alarm.Pitch>,
                                duration:  <C.Response.Alarm.Duration>
                            )
                        endif
                    endfor
    ))

endif
```

Verification Criteria

- The Generation Design specifies mappings that will produce application work products which exhibit the organizational structure specified in the Product Architecture.
- The Generation Design specifies mappings that produce application work products which satisfy the Product Requirements (i.e., the mappings are consistent with Product Requirements variation).
- All variabilities allowed by decisions are properly represented as product variations.
- The effects of variabilities among work products are mutually consistent (i.e., all mappings are consistent).

3. PROCESS DESCRIPTION

3.1 PROCEDURE

The following steps are performed for each work product identified in Process Requirements.

Step 1. Define Work Product Structure

Action Define how decisions affect the structure of the work product.

Input

- Decision Model.
- Product Architecture.

Result Generation Design: Architecture and component mappings.

Heuristics

- It is sufficient to define the work product structure as a mapping from the Decision Model to the organizational structure of the Product Architecture for the work product. The organizational structure defines the components that are required to implement the work product. This mapping determines which elements of the Product Architecture are implemented for a particular Application Model.
- The Product Architecture determines (conditionally and iteratively) how components of each work product are to be derived from Adaptable Components (i.e., the component mapping is provided implicitly by the Product Architecture). The Generation Design should not modify that mapping.
- Represent this mapping in metaprogramming notation associated with each component in the Product Architecture. The mapping is defined in terms of decisions in the Decision Model and determines whether (one or more of) the associated component(s) should be included in the product created for a given Application Model. This mapping is formed by analyzing the Product Architecture and noting conditions that must be true if a particular component is to be included. If a component is always included in the product, metaprogramming notation is not required.
- It is possible that different Adaptable Components will be used to implement a given Product Architecture component, depending on decisions in the Application Model. In this case, a conditional is used to qualify the association (Product Architecture component, Adaptable Component), thus indicating when a particular Adaptable Component is to be used.

Step 2. Define Component Adaptation

Action Define a mapping from the decisions in the Decision Model to adaptations of the Adaptable Components referenced by the component mapping.

Input

- Decision Model.
- Component Design.
- Generation Design: Component mapping.

Result Generation Design: Decision mapping.

Heuristics When a particular Component Design is to be used to implement a particular component in the Product Architecture, the variability of the Adaptable Component (i.e., its parameterization) must be realized in terms of decisions from the Decision Model. Define the value of each parameter (by name) as a derivation from Decision Model decisions.

3.2 Risk Management

Risk The Generation Design will not produce correctly-structured work products.

Implication Application Production will not produce acceptable work products.

Mitigation Derive work product structures from the Generation Design for Application Models of familiar systems, and review the result with experienced engineers to determine whether the result is acceptable.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Product Requirements are incomplete, ambiguous, or inconsistent.

Source Product Requirements Activity.

Response Describe the inadequacies in the Product Requirements. Proceed with Generation Design, and document any assumptions made regarding the inadequate portions of the Product Requirements.

Contingency The Domain Plan cannot be satisfied with available technical capabilities.

Source Domain Management.

Response Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete a Generation Design that satisfies the Domain Plan as closely as possible.

Contingency The practices and procedures specified in the Domain Plan are either ineffective or inefficient.

Source Domain Management.

Response Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

Contingency The Product Architecture is incomplete, ambiguous, or inconsistent.

Source Product Architecture Activity.

Response Describe the inadequacies in the Product Architecture. Proceed with Generation Design, and document any assumptions made regarding the inadequate portions of the Product Architecture.

Contingency The Component Design is incomplete, ambiguous, or inconsistent.

Source Component Design.

Response Describe the inadequacies in the Component Design. Proceed with Generation Design, and document any assumptions made regarding the inadequate portions of the Component Design.

4.2 Feedback from Product Consumers

Contingency Suggestions are made for Generation Design changes to exploit unforeseen opportunities. For example, a situation where substantial software is made available for use in the Domain Implementation that was not available when the Generation Design was completed.

Source

- Generation Implementation Activity.
- Process Support Development Activity.

Response

- Revise the Generation Design.
- Refer to Domain Management for future planning.
- Reject the changes due to conflicts with the Domain Definition.

Contingency The Generation Design does not satisfy the Product Requirements.

Source Verification Activity.

Response Modify the Generation Design to be consistent with the Product Requirements.

Contingency The Generation Design is incomplete, ambiguous, or inconsistent.

Source Generation Implementation Activity.

Response Refine the Generation Design to correct inadequacies.

This page intentionally left blank.

DE.4. DOMAIN IMPLEMENTATION ACTIVITY

1. GETTING STARTED

Domain Implementation is an activity of Domain Engineering for the implementation of product and process support for Application Engineering projects in a business area. The Domain Implementation must satisfy the Domain Specification created by Domain Analysis. Product support consists of a set of production procedures and associated Adaptable Components that can be used to create standardized work products for members of the product family. Process support consists of the automation, procedures, and documentation that define a standard Application Engineering process.

1.1 Objectives

The objectives of the Domain Implementation Activity are:

- To create a set of Generation Procedures and Adaptable Components as specified in the Product Design.
- To create a standardized Application Engineering process as specified in the Process Requirements.

1.2 Required Information

The Domain Implementation Activity requires the following information:

- Domain Development Plan
- Domain knowledge
- Domain Specification
- Existing components

1.3 Experience and Expertise Needs

The Domain Implementation Activity requires the following expertise:

- Understanding of the technologies for creation, adaptation, and composition of parts into systems and the verification of such parts.
- Expertise in the documentation and automated support of Application Engineering processes.
- Understanding of how systems in the domain are built and sufficient expertise to create and document the software for these systems.

2. PRODUCT DESCRIPTION

The Domain Implementation consists of two parts:

- Product Implementation (see Section DE.4.1)
- Process Support (see Section DE.4.2 in future releases)

3. PROCESS DESCRIPTION

The Domain Implementation Activity consists of two subactivities:

- Product Implementation Activity (see Section DE.4.1): An activity for implementing the Generation Procedures and Adaptable Components required by the Product Design.
- Process Support Development Activity (see Section DE.4.2): An activity for creating and integrating documentation, procedures, and mechanisms for the standardization of an Application Engineering process. This activity is for implementing an Application Engineering process, including access to implemented Adaptable Components, that satisfies the Process Requirements.

The information flow among these activities is shown in Figure DE.4-1. The Domain Verification Activity of Domain Analysis performs an independent evaluation of the products of the Domain Implementation to determine compliance with the Domain Specification.

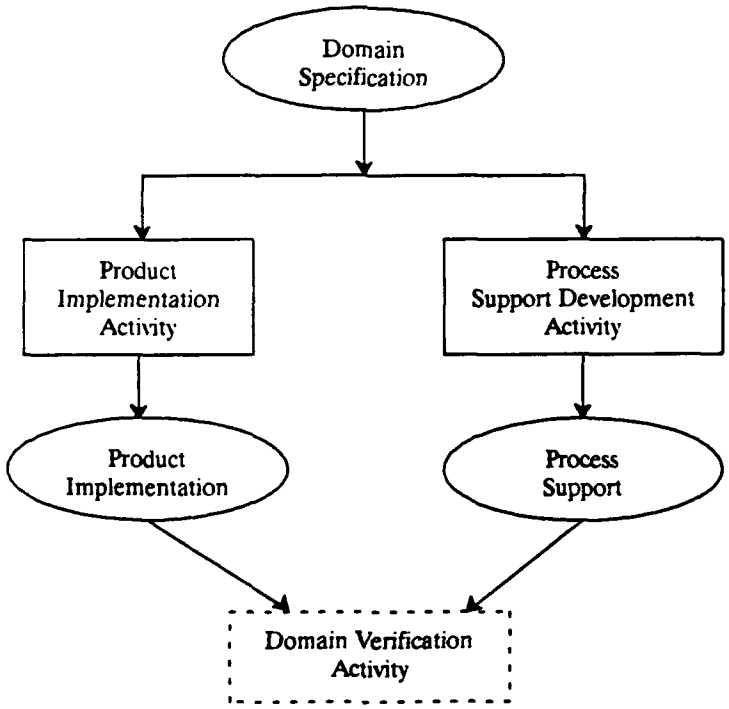


Figure DE.4-1. Domain Implementation Process

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback To Information Sources

Contingency The Domain Specification is incomplete, ambiguous, or inconsistent.

Source Domain Analysis Activity.

Response Describe how the Domain Specification is inadequate and suggest how it may be modified. Proceed with Domain Implementation as far as possible with the current Domain Specification.

Contingency Unforeseen opportunities arise that cannot be exploited given the current Domain Specification, e.g., a situation where substantial software is made available for use in the Domain Implementation that was not available when the Domain Specification was completed.

Source Domain Analysis Activity.

Response Document the opportunities and the required changes to the Domain Specification.

4.2 Feedback from Product Consumers

Contingency The Domain Implementation does not satisfy the Domain Specification.

Source Domain Analysis Activity.

Response Request clarification of the intent of the Domain Specification if necessary. Modify the Domain Implementation to satisfy the Domain Specification.

Contingency The support for the Application Engineering process is inefficient.

Source Application Engineering.

Response Revise the Application Engineering-process support based on Application Engineering experience.

This page intentionally left blank.

DE.4.1. PRODUCT IMPLEMENTATION ACTIVITY

1. GETTING STARTED

The Product Implementation Activity is the activity of the Domain Implementation Activity for creating a Product Implementation. A Product Implementation is an adaptable implementation of application software and associated work products for the members of a product family. A Product Implementation consists of Adaptable Components (e.g., code, documentation, and support for verification/validation) and procedures for selecting, adapting, and composing these components to create work products in accordance with an Application Model that describes the product.

1.1 Objectives

The objective of the Product Implementation Activity is to implement the Product Design. This implementation is used by Application Engineers in Application Production to generate required work products for systems in the domain.

1.2 Required Information

The Product Implementation Activity requires the following information:

- Domain knowledge
- Product Requirements
- Product Design
- Decision Model
- Existing components

1.3 Expertise Needs

The Product Implementation Activity requires the following expertise:

- Understanding of the design method used in specifying the Product Design.
- Understanding of existing systems in the domain, including how they are built and what are their components and architectures.
- Understanding of target language and platform capabilities.
- Understanding of the technologies for adaptation and composition of components into systems.

2. PRODUCT DESCRIPTION

Name Product Implementation.

Purpose A Product Implementation is an adaptable implementation of a product family. An Application Engineer must be able to construct particular instances of work products for systems in the family by adapting the Product Implementation mechanically based on the decisions in an Application Model. The Product Design specifies how these adaptations derive from the Decision Model and how Adaptable Components are to be used in constructing work products for a system.

Content A Product Implementation consists of the following parts:

- Adaptable Components (see Section DE.4.1.1): An implementation of a set of Component Designs. Adaptable Components include software, documentation, and verification/validation components that are adaptable based on decisions in the Decision Model. Collectively, these Adaptable Components form all of the necessary components for constructing, documenting, verifying/validating, and delivering a system.

- Generation Procedure (see Section DE.4.1.2): An implementation of a Generation Design for selecting, adapting, and composing Adaptable Components into deliverable work products that satisfy an Application Model.

Verification Criteria The Product Implementation correctly constructs existing or envisioned systems from the domain.

3. PROCESS DESCRIPTION

The Product Implementation Activity consists of two subactivities:

- *Component Implementation Activity* (see Section DE.4.1.1), which defines how to implement Adaptable Components.

- *Generation Implementation Activity* (see Section DE.4.1.2), which defines how to implement Generation Procedures.

3.1 Risk Management

Risk The Product Implementation will be inconsistent with Product Requirements.

Implication System work products will be generated that do not satisfy the Product Requirements.

Mitigation Review the Domain Specification with domain analysts to clarify their intent when uncertainties arise. Review the Domain Implementation with other experienced engineers to identify omissions and inconsistencies. Derive test work products based on knowledge of existing or anticipated systems for review with experienced engineers.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Domain Specification is incomplete, ambiguous, or inconsistent.

Source Domain Specification Activity.

Response Describe how the Domain Specification is inadequate, and suggest how it may be modified. Proceed with Product Implementation as far as possible with the current Domain Specification.

Contingency Unforeseen opportunities arise that cannot be exploited given the current Domain Specification, e.g., a situation where substantial software is made available for use in the Domain Implementation that was not available when the Domain Specification was completed.

Source Domain Specification Activity.

Response Document the opportunities and the required changes to the Domain Specification.

4.2 Feedback From Product Consumers

Contingency The Product Implementation does not satisfy the Domain Specification.

Source Verification Activity.

Response Request clarification of the intent of the Domain Specification if necessary. Modify the Product Implementation to satisfy the Domain Specification.

Contingency The support for the Application Engineering process is inefficient.

Source Application Engineering.

Response Revise the Application Engineering process support based on Application Engineering experience.

This page intentionally left blank.

DE.4.1.1. COMPONENT IMPLEMENTATION ACTIVITY

1. GETTING STARTED

Component Implementation is an activity of the Product Implementation Activity for creating an Adaptable Component. A component may be a software, documentation, or verification/validation artifact. An Adaptable Component is a representation of a family of such components that satisfies a Component Design (i.e., is adaptable to specified variations). The variability of an Adaptable Component supports extracting concrete components to be used in creating work products for members of a product family.

1.1 Objectives

The objective of the Component Implementation Activity is to implement an Adaptable Component that satisfies a Component Design, consistent with relevant aspects of the Product Architecture and Product Requirements.

1.2 Required Information

The Component Implementation Activity requires the following information:

- Product Requirements
- Product Architecture
- Component Design

1.3 Expertise Needs

The Component Implementation Activity requires the following expertise:

- Understanding of how the capabilities required of the components in a particular family are designed, implemented, and verified.
- Understanding of the technologies for creating and using Adaptable Components.
- Knowledge of the design and implementation of existing systems in the domain.

2. PRODUCT DESCRIPTION

Name Adaptable Component.

Purpose An Adaptable Component is a software, documentation, and verification/validation component that is adaptable based on variations specified in the Component Design.

Content An Adaptable Component is an implementation of a family of components. This family is defined by a Component Design, with support by portions of the Product Requirements and Product Architecture. The Component Design characterizes an Adaptable Component by specifying the permissible adaptations of the component, along with the desired characteristics of its implementation. The Product Architecture also imposes constraints on interactions among components and describes dependencies that exist between them. The Product Requirements define requirements on the system as a whole, parts of which may be assigned by the design as the responsibility of a particular component.

Form and Structure An Adaptable Component is uniquely named and consists of two parts: an adaptability interface and a body.

A component family is characterized by a set of common capabilities and variations in those capabilities. The adaptability interface is a specification of a set of adaptation parameters that provide for the characterization and extraction of a particular instance of a component family.

The body is the sum of the potential implementations of all of the components in the family. The term "potential" is used because the parameters are sufficient to select any component family instance uniquely, but the particular implementation either may not be available or may be extracted from a representation of the family or relevant subfamily. This varies with the mechanism used for implementing adaptability in the Adaptable Component. There are three types of mechanisms for implementing an Adaptable Component: physical separation, generics, and metaprogramming. These may be used in combination to implement a particular family of components.

The Booch Ada stack packages (Booch, 1987) are an example of an Adaptable Component. There is a unifying concept of what a stack is and how it works. Different stack components are extracted based on decisions such as:

- Type: The type of data that can be put into the stack.
- Iteration: Whether an indexing mechanism should be available for moving forward and backward through the stack in addition to simply pushing elements onto and popping elements off of the top of the stack.
- Garbage collection: Whether the stack should manage unused stack space dynamically for later use.
- Bounding: Whether the stack should be bounded in length.

The Booch packages physically implement 26 different types of stacks. This physical separation approach has the advantage of being simple. If a family has 10 instances, there are 10 implementations and each can be written and verified independently. It does not take advantage of similarities among the instances, however, nor does it make explicit how variabilities determine the content of each instance.

Ada provides generic packages as a standard facility that you can use to implement a code Adaptable Component whose instances differ only in the values of well-defined placeholders that can be substituted at compile time. The "type" variability of stack packages may be represented using a generic package. The placeholders are parameters defined in the adaptability interface of the Adaptable Component. This approach has the advantage of representing variabilities for the family more compactly and uniformly; however, only a simple form of parameter substitution is supported.

A metaprogramming approach, such as that described in (Software Productivity Consortium 1991b), uses a preprocessing mechanism to extract a concrete component from an Adaptable Component. This approach embeds target text fragments of a work product (e.g., code, documentation, verification/validation support) with a superimposed metaprogramming notation. The metaprogramming notation specifies how the target fragments are to be combined and adapted, based on the parameters in the adaptability interface of the Adaptable Component. Typical metaprogramming adaptations include:

- Substitution of parameter values.
- Conditional inclusion of text fragments.
- Repetition of text fragments.
- Definition and in-line instantiation of parameterized fragments.

This approach provides greater flexibility in representing a family compactly but results in more complex descriptions. Since many implementations may be derived from a single description, it is necessary both to manually inspect that description and to extract and verify representative instances.

Volume II, Section CS.1.4 presents examples of Adaptable Components that are implemented using Ada generic and metaprogramming approaches.

Verification Criteria

- The Adaptable Components implement their specifications as defined in the Product Architecture and Component Designs.
- The adaptabilities produce the right variations in concrete components.
- Behaviors or constraints imposed by Product Requirements or Product Architecture on the Adaptable Component are all supported.

3. PROCESS DESCRIPTION

The Component Implementation process consists of all activities necessary for implementing an Adaptable Component to its Component Design specifications, consistent with relevant parts of the Product Architecture and Product Requirements. This involves the design, coding, and verification of the family of software, documentation, or test-support components. It may involve the reengineering of existing components of work products from previously-built systems.

3.1 Procedure

These steps are followed for the Component Implementation Activity.

Step 1. Design the Component's Internal Structure.

Action Create an internal design of the structure and elements of the required Adaptable Component.

Input

- Component Design.
- Product Architecture.
- Existing components.

Result

- Adaptable Component: Internal design.
- *Candidate Components.*

Heuristics

- The Component Design specifies the externally-accessible capabilities and required behavior of the components in the family. The internal design is created to satisfy this specification. The structures of the Product Architecture may impose additional requirements on the internal structure (e.g., for concurrency or level of performance) or define constraints on how other components are used in the implementation.
- Envision how required instance components are to be implemented. Create structures, according to the design method used, that characterize the required implementation. Parameterize each structure, if appropriate, with adaptability parameters that vary that structure as required for different component instances. Consider the required operations of the instances.
- Determine whether a suitable component that approximates one of these desired instances is available from existing systems. Identify and evaluate the quality of each such a component and designate it as a Candidate Component for further use. Determine which Adaptable Component variations are implicitly addressed by this selected Candidate Component.

- Consider whether other Adaptable Components in the Product Architecture can be used to implement all or part of this component family. Components that have complex functionality may be implementable as a composition of instances of other, more primitive components.

- If Candidate Components have been identified, consider starting with the internal designs of those components. Portions of several candidate components may be used collectively to implement the Adaptable Component. These components may implement different variations that will be required for the family. Characterize which instance of the family corresponds to each component (i.e., by its parameter values). Consider whether each design is sufficiently well-engineered and representative of the family, or at the least of a subfamily, to provide substantial leverage in being refined to represent other variations and the family as a whole. Consider that they may represent different subfamilies that should be structured differently. See if their designs can be unified using adaptabilities. Be sure you can still derive each of the components with an acceptable structure.

Step 2. Implement the Component

Action Elaborate the internal design to implement the Adaptable Component.

Input

- Candidate Components.
- Component Design.
- Product Architecture.
- Adaptable Component: Internal design.

Result Adaptable Component.

Heuristics

- In implementing the Adaptable Component, fill in the internal structure with the details of the implementation. Keep in mind how the adaptabilities affect the content of the parts of the structure.

- If suitable Candidate Components were used in creating the internal design, then the implementations of those components can be useful as the starting point in implementing the Adaptable Component.

- If another Adaptable Components is to be used in implementing this Component, determine how the adaptability parameters of this Component can be mapped into the parameters of that Component so that its correct instance is derived for a particular instance of this family.

- If it is not possible to obtain all elements of the Component's implementation from existing Candidate Components or other Adaptable Components, then it is necessary to engineer parts of the Adaptable Component from scratch. These areas should be given much greater thought to ensure that you produce the correct content.

Step 3. Verify the Component

Action Verify that the Adaptable Component satisfies all relevant specifications.

Input

- Adaptable Component.
- Component Design.
- Product Architecture.
- Product Requirements.
- Candidate Components.

Result Verified Adaptable Component.

Heuristics

- The first, most comprehensive way to verify an Adaptable Component is through the rigorous inspection of that component by other experienced engineers. The Component Design, as well as relevant parts of the Product Requirements and Product Architecture, should be verified as being satisfied.
- Another effective way to verify an Adaptable Component is to derive representative instances and to test those instances in a conventional fashion to see if they operate correctly. One part of this activity is the creation of test-case scenarios that can be used in regression testing of the Adaptable Component when it is modified in the future. These scenarios may be made adaptable to the same parameters as the Adaptable Component itself so that a scenario can be derived for a particular test instance.
- A third way to verify an Adaptable Component is to rederive the Candidate Components that influenced the implementation. The original and derived instances can then be compared to see if and how they differ and whether an equivalent result can be produced.
- Use of a Candidate Component may have been based on assurances that the component received with respect to certain desired properties such as correctness, reliability, certification, and trust (in the security sense). Note, however, that modification of the component can invalidate some of these assurances (i.e., certification and trust). It is important to verify that the desired properties are retained when the component is extracted from the resulting Adaptable Component.

3.2 Risk Management

Risk Certain combinations of adaptability are not fully supported in the Adaptable Component.

Implication Work products for some systems will not be derivable using the Component.

Mitigation In verifying the Component, use bounds-coverage techniques to identify a variety of adaptability combinations in deriving test instances.

Risk The effort required to implement all specified adaptabilities for an Adaptable Component is not viable compared to that required to develop concrete components which support a single system development effort.

Implication Only a subfamily of the Adaptable Component will be available for production of systems in the domain.

Mitigation Implement the variations required for the current systems under development. The development of these variations may require less effort than developing all possible variations and can be refined as additional needs arise. The Adaptable Component can be evolved to a completed state over several development iterations of a system or systems.

Risk Too much effort will be required in determining the value of existing components as a basis for the Adaptable Component (e.g., too many components to search through, too labor intensive to look through complicated components, too difficult to determine whether a component is correctly implemented).

Implication Effort to evaluate and reengineer existing components exceeds the effort to create the Adaptable Component from scratch.

Mitigation If it is determined that there are just too many existing components to search through to find a good baseline component, limit the amount of effort dedicated to the search, and use the best approximation that results from the limited search.

If it is too labor intensive to look through complicated components, reduce the number of components that will be reviewed. If the component is overly complicated, it may be beneficial to rely on higher-level documentation (i.e, requirements, high-level design, or testing documentation) of the component as an indicator of the worth of the component. Reviewing documentation on the existing component is likely to take less effort than reviewing code.

If it is too difficult to determine the correctness of the component, then previous testing documentation on the component may need to suffice to determine correctness. Reliance on existing components may be greater if engineers are available who developed, or at least used, the existing components.

Risk Modification of the baseline component may invalidate assurances of quality that the component possessed (e.g., security or certification).

Implication Modification of the trusted component will require that the resulting Adaptable Component must pass once again the tests required for assurance of given properties.

Mitigation If the process of proving assurance of given properties is an expensive or time-consuming process, try to retain the essential nature of the component in the Adaptable Component so that at the end, the exact component can be extracted that existed at the start. Concentrate effort on areas of particular concern. If the given properties are less important for the family as a whole, treat that component as a special case (i.e., a subfamily in its own right).

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Component Design is incomplete, ambiguous, or inconsistent.

Source Component Design Activity.

Response Describe how the Component Design is inadequate, and suggest how it may be modified. Proceed with Component Implementation as far as possible with the current Component Design.

4.2 Feedback from Product Consumers

Contingency The Component Implementation does not satisfy the Component Design.

Source Verification Activity.

Response Request clarification of the intent of the Component Design if necessary. Modify the Component Implementation to satisfy the Component Design.

DE.4.1.2. GENERATION IMPLEMENTATION ACTIVITY

1. GETTING STARTED

Generation Implementation is an activity of the Product Implementation Activity for creating a *Generation Procedure*. A Generation Procedure is a precise description of how to derive application work products consistent with an Application Model. A Generation Procedure may be automated or may take the form of a precise description that application engineers can follow in Application Production to create required work products.

1.1 Objectives

The objective of the Generation Implementation Activity is to create a Generation Procedure as specified by a Generation Design.

1.2 Required Information

The Generation Implementation Activity requires the following information:

- Generation Design

1.3 Expertise Needs

The Generation Implementation Activity requires the following expertise:

- Understanding of the notation used in specifying the Generation Design.
- Understanding of the technologies for adaptation and composition of components into work products.

2. PRODUCT DESCRIPTION

Name Generation Procedure.

Purpose This is a procedure for creating a standardized work product using Adaptable Components. This procedure is either implemented as a product generator or documented as a manual procedure to be followed by application engineers.

Content The Generation Procedure is a procedural description for producing an application work product that satisfies the mappings of a Generation Design (see Section DE.3.2.4.3). It describes how to select appropriate Adaptable Components, how to apply decisions from an Application Model to adapt them, and how to compose them to create the work product in final form.

Form and Structure

A Generation Procedure is a textual description of how to select, adapt, and compose an application work product in accordance with decisions in an Application Model for a member of the product family.

The form of the Generation Procedure differs, depending on whether the procedure is automated or not. The Generation Procedure is either implemented as an automated product generator or documented as a manual procedure to be followed by application engineers. If the manual form is chosen, then the Generation Procedure form is likely to resemble the form of a Generation Design. If the Generation Procedure is implemented in the form of a product generator, however, it will be a conventional software program.

Verification Criteria

- The Generation Procedure can be used to produce application work products that exhibit the organizational structure specified in the Product Architecture.
- The Generation Procedure can be used to produce application work products that satisfy the Product Requirements.
- If a manual form is used, the Generation Procedure clearly describes how system work products are constructed from Adaptable Components based upon decisions contained in an Application Model.

3. PROCESS DESCRIPTION

Perform the appropriate one of the following two steps. The appropriate action depends on whether creation of the corresponding work product is to be manual or automated.

Step 1. Document the Generation Procedure

Action Document a manual Generation Procedure.

Input Generation Design.

Result Generation Procedure.

Heuristics The Generation Design is a precise description of the required Generation Procedure. Document the procedure in a form that is usable by application engineers.

Step 2. Automate the Generation Procedure

Action Implement the Generation Procedure as defined in the Generation Design.

Input Generation Design.

Result Generation Procedure.

Heuristics

- The Generation Design describes, using Adaptable Components, the selection, adaptation, and composition of a work product. Automated support for this procedure requires access to the decisions of an Application Model that describes the required member of the product family. If the Application Model is automated, then the Generation Procedure must access it. If it is not automated, then there must be an automated mechanism for providing the decisions of the Application Model as input to the Generation Procedure.

- The Generation Procedure must be able to retrieve and instantiate required Adaptable Components. If a metaprogramming technology, such as described in the Component Implementation Activity (see Section DE.4.1.1), is used to implement the Adaptable Components, then it is used to instantiate those components. Metaprogramming technology may also be useful in implementing portions of the Generation Procedure itself.

- The creation of an automated Generation Implementation is a software development task. It requires the design of the required program, implementation to that design in a programming language, testing to verify that the resulting program implements the Generation Design correctly, and documentation so that the program can be correctly modified as the Generation Design changes.

- Tools such as the UNIX **make** facility may be useful in automating the procedure for composing adapted components into final work products.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Generation Design is incomplete, ambiguous, or inconsistent.

Source Generation Design Activity.

Response Describe how the Generation Design is inadequate, and suggest how it may be modified. Proceed with Generation Implementation as far as possible with the current Generation Design.

Contingency Unforeseen opportunities arise that cannot be exploited given the current Generation Design, e.g., a situation where substantial software is made available for use in the Generation Implementation that was not available when the Generation Design was completed.

Source Generation Design Activity.

Response Document the opportunities and the required changes to the Generation Design.

4.2 Feedback from Product Consumers

Contingency The Generation Implementation does not satisfy the Generation Design.

Source Verification Activity.

Response Request clarification of the intent of the Generation Design if necessary. Modify the Generation Implementation to satisfy the Generation Design.

Contingency A manual Generation Procedure is difficult to use.

Source Application Engineering Activity.

Response Investigate new forms for conveying the Generation Procedures to the Application Engineers.

PART III: APPLICATION ENGINEERING

This page intentionally left blank.

AE.1. APPLICATION ENGINEERING OVERVIEW

1. GETTING STARTED

Application Engineering is an activity of Synthesis for the creation and support of a product that satisfies specified customer requirements. A product is represented by a set of associated work products that result from analysis of those requirements. Application Engineering is characterized by a comprehensive life-cycle process for the management, analysis, production, and support of the members of a product family. This is similar in purpose to a conventional software development process, but is tailored to the problems and the needs of projects in a particular business area. Such a business-area focus allows for the systematic reuse of standardized work products within and among projects in that business area.

Domain Engineering identifies a set of characteristic decisions for a business area that determine how a product can be tailored to meet particular needs. It also provides standardized work products in a form that supports tailoring to those decisions. Application Engineering concentrates on the analysis of customer requirements to resolve those decisions. The result is a model of a corresponding product that can be evaluated according to those requirements. When a model is found to be acceptable, it is used to drive the generation of tailored work products that implement the model. After the resulting work products are verified to the model, they are delivered to customers for further evaluation and use.

1.1 Objectives

The objectives of Application Engineering are:

- To organize and direct resources for the production and support of a product.
- To understand the needs of customers, balancing concerns of cost versus value, to produce a product that fulfills those needs most effectively.
- To produce software and documentation that support the delivery and use of the product.
- To leverage standardized process and work products for a domain to produce required results effectively, predictably, and profitably.

1.2 Required Information

Application Engineering requires the following information:

- *Application Engineering Process Support*
- Customer requirements

1.3 Experience and Expertise Needs

Application Engineering requires the following expertise:

- Understanding of the problems that the products in the domain are intended to solve and the engineering tradeoffs to be considered in creating a viable solution.
- An ability to understand and interpret customer requirements and restate them in notations appropriate to the domain.
- Experience in the management, production, and delivery of software work products.

2. PRODUCT DESCRIPTION

Application Engineering creates four work products:

- *Project Plan* (see Section AE.2 in future releases), which establishes standard practices and procedures, and defines tasks for incremental development with milestones and resource allocations.
- *Application Model* (see Section AE.3 in future releases), which describes a deliverable system in terms of requirements and engineering decisions.
- *Application Software* (see Section AE.4 in future releases) (code and documentation), which is derived mechanically from the Application Model to provide a capability specified by customer requirements.
- *Delivery Support* (see Section AE.4 in future releases) (such as testing, installation, and training materials), which is derived mechanically to be consistent with the Application Model to support the delivery of Application Software.

The set of Application Software and Delivery Support work products that are delivered to the customer are labeled Deliverables.

3. PROCESS DESCRIPTION

The Application Engineering process defined here (shown in Figure AE.1-1) is prototypical in the sense that an objective of Domain Engineering is to define such a process tailored to the needs of a domain. This Application Engineering process consists of an interaction among four subactivities:

- *Project Management* (see Section AE.2): An activity for planning, monitoring, and controlling project resources to deliver a product.
- Application Modeling (see Section AE.3): An activity for the resolution of decisions that distinguish (and therefore describe) a required product and for the evaluation of that description in terms of customer requirements.
- *Application Production* (see Section AE.4): An activity for the creation of standardized Application Software and Delivery Support work products in accordance with prescribed standards and procedures for the domain.
- *Delivery and Operation Support* (see Section AE.5): An activity for delivering Application Software to customers, supporting its use, and evaluating its effectiveness.

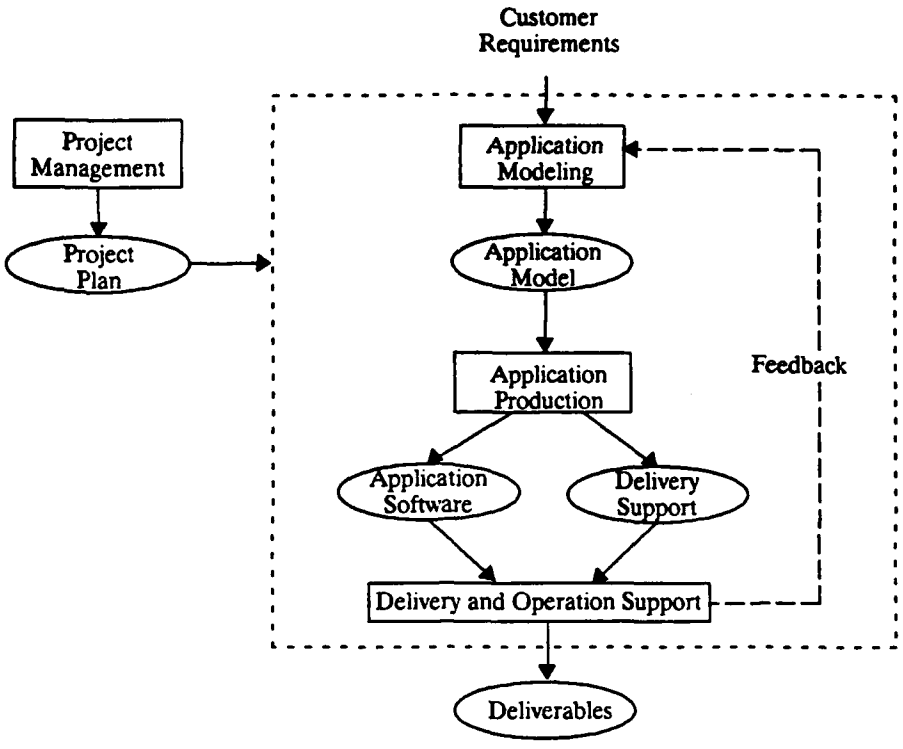


Figure AE.1-1. (Prototypical) Application Engineering Process

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback To Information Sources

Contingency The standardized product family is inadequate to support the needs of particular customers.

Source Domain Engineering.

Response Describe why the standardized product family is deemed inadequate, and suggest how it can be improved.

Contingency The standardized Application Engineering process is inefficient or leads to less-than-ideal results for this project.

Source Domain Engineering.

Response Describe why the standardized Application Engineering process is deemed inefficient or inadequate, and suggest how it can be improved.

4.2 Feedback from Product Consumers

This page intentionally left blank.

APPENDIX A. MATURITY ASSESSMENT AND FUTURE EVOLUTION

This is the first release of the *Synthesis Guidebook*. As a whole, the Consortium considers it to be at the exploratory level in the following maturity scheme:

- Exploratory: Many sections are incomplete or missing; incomplete sections are limited in detail; Consortium assistance is needed for effective use.
- Developmental: All sections are present but some are incomplete; complete sections have been used on at least one pilot project or case study and are normally usable with only limited Consortium assistance.
- Functional: All sections are complete, have been independently validated in use on at least two pilot projects or case studies, and are generally usable without Consortium assistance.
- Production: All sections are complete, validated, and constitute a mature representation of organizational policies and concerns.

To advance beyond functional-level maturity, the guidebook must be refined to meet particular standards for production use in each member company. The Consortium can only advise member company organizations in refining the guidebook from functional-level to production-level maturity. Once the Guidebook reaches production-level maturity, quality improvements can continue based on member company experience with using it.

The current plan is to reach functional-level maturity for the entire guidebook by the end of 1994. This goal assumes pilot project participation by member companies. Such participation is essential to improving the depth and quality of the guidebook.

At this stage, we consider the guidebook to be usable in the following ways:

- On pilot projects.
- Under the guidance of a technologist, acting as a transfer agent, who can interpret, elaborate, and fill in guidebook material sufficiently for other participants to be effective in performing Synthesis.
- With less mature line managers and engineers as primary participants, for essential domain expertise and for training.
- With Consortium assistance and review, making substantive contributions in the case of exploratory-level sections, but limited to advising the technologist in the case of developmental-level sections and to independent review in the case of functional-level sections.

The maturity scheme described above also applies to the individual sections of the guidebook. The current maturity of each section is shown in Table A-1.

Table A-1. Maturity Scheme for *Synthesis Guidebook Sections*

SECTION	MATURITY
OV. Overview	Exploratory
DE.1. Domain Engineering Overview	Exploratory
DE.2. Domain Management	Exploratory
DE.3. Domain Analysis	Exploratory
DE.3.1. Domain Definition Activity	Exploratory
DE.3.2. Domain Specification Activity	Exploratory
DE.3.2.1. Decision Model Activity	Developmental
DE.3.2.2. Product Requirements Activity	Developmental
DE.3.2.3. Process Requirements Activity	Exploratory
DE.3.2.4. Product Design Activity	Exploratory
DE.3.3. Domain Verification Activity	Omitted
DE.4. Domain Implementation Activity	Exploratory
DE.4.1. Product Implementation Activity	Exploratory
DE.4.1.1. Component Implementation Activity	Developmental
DE.4.1.2. Generation Implementation Activity	Exploratory
DE.4.2. Process Support Development Activity	Omitted
DE.5. Project Support Activity	Omitted
AE.1. Application Engineering Overview	Exploratory
AE.2.-AE.5.	Omitted
AT. Advanced Topics	Omitted

APPENDIX B. REFERENCE MODEL CORRESPONDENCE

Tables B-1 through B-4 indicate how the correspondence between terminology used in this guidebook corresponds to that defined in the *Synthesis Methodology Reference Model* (Campbell 1990a). Indentation indicates the components of a product or activity.

Table B-1. Domain Engineering Products

Reference Model	Guidebook
domain engineering plans	Domain Plan
(none)	Domain Evolution Plan
(none)	Domain Development Plan
domain definition	Domain Definition
domain description	Domain Synopsis
glossary	Domain Glossary
assumptions	Domain Assumptions
domain status	Domain Status
viability analysis	Viability Analysis
domain model	Domain Specification
conceptual framework	(none)
representation	Decision Model
presentation	Process Requirements
semantics	Product Requirements
(none)	Product Design
system composition mapping	Generation Design
(none)	Product Architecture
reuse architecture	Component Design
domain implementation	Domain Implementation
(none)	Product Implementation
candidate components	Candidate Components
reuse library	Adaptable Components
(none)	Generation Procedure
(none)	Process Support
application engineering environment	Application Engineering Environment

Table B-2. Domain Engineering Activities

Reference Model	Guidebook
domain management	Domain Management Activity
domain analysis	Domain Analysis Activity
domain qualification	Domain Definition Activity
domain model formulation	Domain Specification Activity
conceptual framework design	(none)
representation design	Decision Model Activity
presentation design	Process Requirements Activity
semantics design	Product Requirements Activity
system composition mapping design	Product Design Activity
(none)	Generation Design Activity
(none)	Product Architecture Activity
reuse architecture design	Component Design Activity
verification	Domain Verification Activity
domain implementation	Domain Implementation Activity
(none)	Product Implementation Activity
reuse engineering	Component Implementation Activity
component qualification	(none)
(none)	Generation Implementation Activity
environment engineering	Process Support Development Activity
(none)	Project Support Activity

Table B-3. Application Engineering Products

Reference Model	Guidebook
project plan	Project Plan
application model	Application Model
application software	Application Software
(none)	Special Components
integration support	Delivery Support

Table B-4. Application Engineering Activities

Reference Model	**Guidebook**
project management	Project Management Activity
application modeling	Application Modeling Activity
specification	Specification Activity
validation	Validation Activity
assessment	Assessment Activity
(none)	Application Production Activity
generation	Generation Activity
application-specific implementation	Special Component Implementation Activity
application integration	Delivery and Operation Support Activity

This page intentionally left blank.

GLOSSARY

Abstract component	A family of components characterized by a defining abstraction and the decisions that are needed to distinguish among the members of the family (or to extract a concrete component).
Abstraction	A concept that denotes the common properties of a family; a description of a set of things that applies equally well to any one of those things.
Activity	A work assignment organized to produce and modify a set of related work products and/or to provide other activities with feedback on their work products.
Adaptable Component	A Domain Engineering work product that implements a Component Design. See Abstract component.
Application	The hardware, software, and manual procedures that characterize a system.
Application Engineering	An iterative process for the design and development of a product that satisfies specified customer requirements. Its work products are an Application Model and Application Software. See Application Modeling (Activity), Application Production (Activity), Delivery and Operation Support (Activity), and Project Management (Activity).
Application Engineering Environment	An automated framework that supports a prescribed Application Engineering process. See Process Requirements and Process Support and Infrastructure.
Application Engineering Process Support	A Domain Implementation, from the perspective of an Application Engineering project.
Application Model	A set of requirements and engineering decisions, as specified by a Decision Model, that (partially) determine an instance of a family of systems. See Application Modeling Notation.

Application Modeling (Activity)	An Application Engineering activity that produces a validated Application Model sufficient to derive production-quality Application Software that satisfies customer requirements. See Specification (Activity), Validation (Activity), and Assessment (Activity).
Application Modeling Notation	A notation for expressing an Application Model such that a complete Application Model is sufficient to distinguish uniquely any system of a domain. See Process Requirements.
Application Production (Activity)	An Application Engineering activity that produces Application Software, as specified by an Application Model, and Delivery Support. See Generation (Activity) and Special Components Implementation (Activity).
Application Software	Software artifacts, including code and documentation, produced by the Application Production activity to satisfy customer requirements.
Architecture	A set of design structures that characterize a system and each associated artifact (i.e., work products).
Assessment (Activity)	An Application Modeling activity that produces analyses of the degree to which alternative Application Models satisfy the operational (e.g., performance, reliability) and product quality requirements of the customer.
Business-area knowledge	Information that characterizes the market for a domain including: • Current and future customer and contract profiles. • Projected growth in contracts (or sales). • Value and marketability of features. • Market analyses.
Business-area organization	An organization whose mission is the production and delivery of products for customers in a specified business area.
Business objectives	The needs of a business-area organization that determine the scope and extent of a domain.

Candidate Components	A set of previously-built components that have been judged as qualified for potential use as raw material in the engineering of Adaptable Components.
Commonality	A characteristic of a domain that corresponds to a similarity among members of the associated family of systems. See Variability.
Component	A work-product fragment whose production is a managed work assignment. See Abstract component.
Component Design	The element of a Product Design that defines the design of a component identified in a Product Architecture.
Component Design (Activity)	The Domain Analysis activity that creates a Component Design.
Component Implementation (Activity)	The Domain Implementation activity that creates Adaptable Components, as specified by a Component Design.
Constraint	A limitation on the decision-making process that application engineers follow in specifying a system. See Structural constraint and Dependency constraint.
Customer requirements	A description of the capabilities, as understood by customers, required of a system and any constraints on the engineering of that system.
Decision	A question that differentiates among the members of a family (e.g., of systems or components) and an associated characterization of allowable answers.
Decision group	A logical grouping of decisions in the Application Modeling Notation that allows the application engineer to separate concerns when describing a system.
Decision Model	The element of a Domain Specification that defines the abstract form (concepts, decisions, and structure) of an Application Modeling Notation.
Decision Model (Activity)	The Domain Analysis activity that creates a Decision Model.
Deliverable	An Application Engineering work product that is required to be delivered to the customer.

Delivery and Operation Support (Activity)	The Application Engineering activity that delivers Application Software to customers and supports its use.
Delivery Support	Software artifacts produced by the Application Production activity to support the Delivery and Operation Support activity for delivery of Application Software to customers.
Dependency constraint	A relationship specifying how decisions made by an application engineer limit subsequent decisions.
Derived requirements	Requirements that indicate characteristics specific to particular systems in the domain based on the decisions that an application engineer expresses in an application model.
Design structure	A set of relationships among a set of components that represent some characteristic of the aggregate. Examples for a program are dependency structures that define a program's static structure and process structures that define its dynamic behavior.
Document	A documentation component, including textual and graphical artifacts.
Domain	(1) A coherent business area. (2) The denotation of a product family for which Application Engineering Process Support is needed.
Domain Analysis (Activity)	The study and formalization of domain knowledge in a Domain Definition and a Domain Specification. The expertise in a business area is formalized to create standards for problem descriptions and corresponding solutions. See Domain Definition (Activity), Domain Specification (Activity), and Domain Verification (Activity).
Domain Assumptions	The element of a Domain Definition that defines the guiding assumptions and justifications of domain scope and extent.
Domain Definition	An informal description of the scope, extent, and justification for a domain. See Domain Synopsis, Domain Glossary, Domain Assumptions, and Domain Status.

Domain Definition (Activity)	A Domain Analysis activity that creates a Domain Definition.
Domain Engineering	An iterative process for the design and development of (1) a product family and (2) an Application Engineering process for producing members of that family. See Domain Management (Activity), Domain Analysis (Activity), Domain Implementation (Activity), and Project Support (Activity).
Domain evolution	Revision of a Domain Definition, Domain Specification, and associated Application Engineering Process Support to reflect changes in domain scope.
Domain Glossary	The element of a Domain Definition that defines the terminology of a domain.
Domain Implementation	A Product Implementation and Process Support that satisfies a Domain Specification.
Domain Implementation (Activity)	A Domain Engineering activity that creates support for Application Engineering projects in the form of a Domain Implementation. See Product Implementation (Activity) and Process Support Development (Activity).
Domain knowledge	Knowledge and expertise characteristic to a domain: • Relevant scientific theory and engineering practice. • Capabilities and uses of existing systems. • Past system development and maintenance experience and work products. • Potential developments in related or supporting technology. • Potential changes in customer needs.
Domain Management (Activity)	A Domain Engineering activity that plans, monitors, and controls the activities and resources of a Domain Engineering organization and which coordinates domain development and evolution with client Application Engineering projects.

Domain objectives	An element of the Domain Plan that defines the goals of domain development.
Domain Plan	Schedules, budgets, assignments, and progress evaluations for the management of a Domain Engineering organization.
Domain Specification	A specification of a standardized Application Engineering process and product family for a domain. See Decision Model, Product (Family) Requirements, Process Requirements, and Product (Family) Design.
Domain Specification (Activity)	The Domain Analysis activity that creates a Domain Specification.
Domain Status	The element of a Domain Definition that specifies the current scope, extent, and viability of a domain relative to its objectives.
Domain Synopsis	The element of a Domain Definition that is an informal description of a domain.
Domain Verification (Activity)	The Domain Analysis activity that performs an independent evaluation of the products of Domain Implementation to determine compliance with a Domain Specification.
Family	A set of things that have enough in common that it pays to consider their common characteristics before noting specific properties of instances.
Feasibility	The degree to which an objective is amenable to solution with predictable resources and risk.
Feedback	Information communicated by the consumer of a work product to its producer regarding issues in the correctness, quality, and viability of the product.
Generation (Activity)	The Application Production activity that applies a Generation Procedure to an Application Model, Adaptable Components, and Special Components to produce software work products.
Generation Design	The element of a Product Design that specifies a Generation Procedure (i.e., the mapping from a Decision Model and Product Architecture to work products for an application).

Generation Design (Activity)	The Domain Implementation activity that creates a Generation Design.
Generation Implementation (Activity)	The Domain Implementation activity that creates a Generation Procedure.
Generation Procedure	The definition of a procedure for the selection, adaptation, and composition of components to create a work product.
Implicit requirements	Requirements that indicate characteristics that are common to all systems in a domain. (*COMMENT*: These are referred to as implicit because they are implicit to an Application Model [i.e., there are no decisions in the Decision Model that affect them].)
Infrastructure	Those mechanisms or attributes of an Application Engineering Environment that are not determined by the Domain Specification. See Process Support Development (Activity).
Instantiation	The application of specified decisions to an abstract component to extract a concrete component (i.e., to select a particular member of a family of components). See Specialization.
Iterative process	A process in which completion occurs only after repetition of producing and using activities results in refined work products.
Metaprogram	A description of an abstract component.
Metaprogramming	A method for creating abstract components and extracting concrete components from them. See Instantiation.
Metaprogramming notation	A notation for defining and instantiating metaprograms.
Method	A precise, systematic technique to be used in performing an activity.
Methodology	A definition of a set of work products and a set of activities structured into a life-cycle process to produce and modify those work products.
Module	A software component that consists of design, code, documentation, and test artifacts.

Process	(1) A partial ordering of the activities that comprise a methodology or an activity of a methodology. (2) An ordered set of events that accomplishes some task required of a system.
Process Requirements	The element of a Domain Specification that defines an Application Engineering process and concrete forms (syntax) of the corresponding Application Modeling Notation.
Process Requirements (Activity)	The Domain Analysis activity that creates Process Requirements.
Process Support	Standards and procedures, in the form of documents and supporting automation, that institute a standard Application Engineering process, as specified by a Process Requirements. See Application Engineering Environment.
Process Support Development (Activity)	The Domain Implementation activity that creates Process Support.
Product	An application and all associated deliverables and supporting work products.
Product Architecture	See Product (Family) Architecture.
Product Family	A family of products.
Product (Family) Architecture	The element of a Product Design that is a specification of the (adaptable) architecture of the products for a system (possibly as a set of components).
Product Architecture (Activity)	The Domain Analysis activity that creates a Product Architecture.
Product (Family) Design	The element of a Domain Specification that defines how an Application Model which satisfies the Decision Model determines the structure and content of Application Software and Delivery Support. This includes the criteria by which components are selected and adapted to create fragments which are then composed into complete work products.
Product Design (Activity)	The Domain Analysis activity that creates a Product Design.

Product Design	See Product (Family) Design.
Product (Family) Implementation	The implementation of a Product Design as sets of Adaptable Components and Generation Procedures.
Product Implementation	See Product (Family) Implementation.
Product Implementation (Activity)	The Domain Implementation activity that creates a Product Implementation. See Component Implementation (Activity) and Generation Implementation (Activity).
Product (Family) Requirements	The element of a Domain Specification that defines the requirements of systems in a domain relative to a Decision Model.
Product Requirements	See Product (Family) Requirements.
Product Requirements (Activity)	The Domain Analysis activity that creates a Product Requirements.
Program	An aggregation of software components that, when integrated with hardware, operates as a unit.
Project	An organization whose mission is the production and delivery of an application that satisfies specified customer requirements.
Project Management (Activity)	The Application Engineering activity that plans, monitors, and controls Application Engineering process execution and provides feedback to Domain Engineering on desired product family and application engineering process modifications.
Project Plan	Schedules, budgets, assignments, and status evaluations for the management of an Application Engineering project.
Project Support (Activity)	The Domain Engineering activity that delivers Process Support to application projects and supports its use.
Risk	Uncertainty in an undertaking that may cause reduced benefit or unexpected costs.
Special Component Implementation (Activity)	An Application Production activity for the creation or modification of Special Components.

Special Component	A component that satisfies requirements outside the scope of any Adaptable Component. Equivalently, a component that cannot be specified with or generated from the Application Modeling Notation but that is needed to satisfy customer requirements.
Specialization	The application of a subset of specified decisions to an abstract component to derive another abstract component that characterizes a subfamily of the initial component. See Instantiation.
Specification	A complete, precise description of the verifiable properties required of a work product.
Specification (Activity)	An Application Modeling activity that analyzes customer needs to produce an application model. The Application Model expresses requirements and engineering decisions that describe a system intended to satisfy those needs.
Structural constraint	A constraint that limits the number of instances of a decision group in a valid Application Model
Subdomain	The denotation of a subfamily of systems for which a corresponding domain denotes the larger family.
Subfamily	A subset of the members of a family that have some set of common characteristics not shared by any members of the family outside that subset. See subdomain.
Synthesis	A methodology for the construction of software systems as instances of a family of systems that have similar descriptions. Its primary distinguishing features are: • Formalization of domains as families of systems that share many common features, but which also vary in well-defined ways.

Synthesis (continued)	• System building reduced to resolution of requirements and engineering decisions that represent the variations characteristic of a domain. • Reuse of software artifacts through mechanical adaptation of components to satisfy requirements and engineering decisions. • Model-based analyses of described systems to help understand the implications of system-building decisions and evaluate alternatives.
System	A collection of programs, hardware, and people that operate together to accomplish a mission.
Test scenario	A test component, which includes test procedure and test data artifacts.
Validation (Activity)	The Application Modeling activity that produces analyses of the degree to which alternative Application Models satisfy the functional requirements of the customer.
Variability	A characteristic of a domain that corresponds to features that distinguish among members of the associated family of systems. See Commonality.
Viability	The degree to which benefits of a feasible undertaking dominate the costs of its performance, taking into consideration risk-induced uncertainties.
Viability Analysis	The element of a Domain Status that evaluates the viability of the (current and future) scope and extent of a domain.
Work product	Any configuration-managed artifact required by a methodology.

This page intentionally left blank.

REFERENCES

Balzer, Robert, and Neil Goldman 1979 — *Principles of Good Software Specification and their Implications for Specification Languages.* USC/Information Sciences Institute.

Booch, Grady 1987 — *Software Components with Ada.* Menlo Park, California: Benjamin-Cummings.

Borgida, Alexander 1985 — Features of Languages for the Development of Information Systems at the Conceptual Level. *IEEE Software* 1:63–72.

Burkhard, Neil, Jeff Facemire, James Kirby, Jr., and James O'Connor 1990 — *Applying Synthesis in the Design Composer Domain,* SPC-90078-MC. Herndon, Virginia: Software Productivity Consortium.

Campbell, Grady H., Jr. 1989 — *Abstraction-Based Reuse Repositories,* REUSE_REPOSITORIES-89041-N. Herndon, Virginia: Software Productivity Consortium.

1990a — *Synthesis Methodology Reference Model,* SYNTHESIS_REF_MODEL-90047-MC. Herndon, Virginia: Software Productivity Consortium.

1990b — *Synthesis Program—A Risk Analysis,* SYNTHESIS_RISK_ANALY-90035-MC. Herndon, Virginia: Software Productivity Consortium.

Campbell, Grady H., Jr., Stuart R. Faulk, and David M. Weiss 1990 — *Introduction to Synthesis,* INTRO_SYNTHESIS_PROCESS-90019-N. Herndon, Virginia: Software Productivity Consortium.

Cruickshank, Robert D., and John E. Gaffney, Jr. 1990 — *Synthesis Economic Analysis and Reuse Economic Model Improvements,* SYNTHESIS_ECON_MODEL-90020-MC. Herndon, Virginia: Software Productivity Consortium.

1991 — *The Economics of Software Reuse,* SPC-91128-MC. Herndon, Virginia: Software Productivity Consortium.

Davis, Alan M., Edward H. Bersoff, and Edward R. Comer 1988 — A Strategy for Comparing Alternative Software Development Life Cycle Models. *IEEE Transactions on Software Engineering* SE-14:1453–61.

Dijkstra, E.W.
1972
Notes on Structured Programming. *Structured Programming*, O.J. Dahl, E.W. Dijkstra, and C.A.R. Hoare, Eds. London: Academic Press:1–82.

Faulk, Stuart, James Kirby, Jr., Skip Osbourne, D. Douglas Smith, Steven Wartik, John Brackett, and Paul T. Ward
1991
The Consortium Requirements Engineering Method, SPC-91140-MC. Herndon, Virginia: Software Productivity Consortium.

Heninger, Kathryn L.
1980
Specifying Software Requirements for Complex Systems: New Techniques and Their Application. *IEEE Transactions on Software Engineering*, SE-6, 2–13.

Heninger, Kathryn, J. Kallander, David L. Parnas, and John Shore
1978
Software Requirements for the A-7E Aircraft. Memorandum Report 3876. Washington, D.C.: Naval Research Laboratory.

Jenkins, Burvin, and Jeff Facemire
1991
An Application Generator Tool Evaluation Method, SPC-91110-MC. Herndon, Virginia: Software Productivity Consortium.

Kent, William
1978
Data and Reality. North Holland, Amsterdam.

Parnas, David L.
1976
On the Design and Development of Program Families. *IEEE Transactions on Software Engineering* SE-2:1–9.

Software Productivity Consortium
1991a
ADARTS Guidebook, SPC-91104-MC. Herndon, Virginia: Software Productivity Consortium.

1991b
TRF2 Metaprogramming Tool User Guide, SPC-91132-MC. Herndon, Virginia: Software Productivity Consortium.

1991c
Evolutionary Spiral Process Guidebook, SPC-91076-MC. Herndon, Virginia: Software Productivity Consortium.

1991d
Systematic Reuse: The Competitive Edge, SPC-91047-N. Herndon, Virginia: Software Productivity Consortium.

1991e
Synthesis Seminar Notebook, SPC-91183-MC. Herndon, Virginia: Software Productivity Consortium.

Wartik, Steve, Neil Burkhard, James O'Connor, Jeff Facemire, Grady Campbell, Guy Cox, Burvin Jenkins, and Yoav Intrator 1990 — *Synthesis Status Report: Fourth Quarter 1990*, SPC-90110-MC. Herndon, Virginia: Software Productivity Consortium.

Weiss, David M. 1990 — *Synthesis Operational Scenarios*, SYNTHESIS_OP_SCENARIOS-90038-MC. Herndon, Virginia: Software Productivity Consortium.

Williams, Roger B. 1990a — *Member Company Domain Characterization for Synthesis*, SYNTHESIS_DOMAIN_CHARAC-90049-MC. Herndon, Virginia: Software Productivity Consortium.

1990b — *Synthesis Transition Strategies*, SYNTHESIS_TRANSITION-90032-MC. Herndon, Virginia: Software Productivity Consortium.

Winograd, Terry 1979 — Beyond Programming Languages. *Communications of the ACM*, 22: 391–401.

This page intentionally left blank.

PLEASE COMPLETE THIS REGISTRATION FORM.

FOLD, TAPE, AND MAIL TO RECEIVE UPDATES TO THE SYNTHESIS GUIDEBOOK.

We are working continually to improve the Synthesis Guidebook and its application. Please help by completing the questions (on the reverse side) to tell us how you plan to use it. Thank you for your time and help.

FOLD HERE

FOLD HERE

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 7702 HERNDON VA

POSTAGE WILL BE PAID BY ADDRESSEE

TECHNOLOGY TRANSFER CLEARINGHOUSE
SOFTWARE PRODUCTIVITY CONSORTIUM, INC.
SPC BUILDING
2214 ROCK HILL RD
HERNDON VA 22070-9858

How did you learn of the Synthesis Guidebook?

- ☐ Received info from company distribution point
- ☐ Heard a Consortium technical presentation
- ☐ Attended a Consortium seminar or workshop
- ☐ Read about it in the Consortium *Quarterly*
- ☐ Saw information on a bulletin board
- ☐ Heard about it at a (non-Consortium) conference
- ☐ A fellow engineer told me
- ☐ Other _____

In what kind of effort will Synthesis be used? (Check all that apply)

Contract	IR&D	Bid & Proposal	Evaluation	(Other) Overhead
☐	☐	☐	☐	☐

How soon will you use Synthesis?

Immediately	Within 3 months	Within 6 months	When our proposal wins	Waiting on the RFP
☐	☐	☐	☐	☐

Estimated number of Synthesis Guidebook users

1 to 3	4 to 6	7 to 10	11 to 20	more than 20
☐	☐	☐	☐	☐

In what application domain will you use Synthesis?

_____

_____

SYNTHESIS GUIDEBOOK REGISTRATION FORM

(Please fill out to receive updates and other information)

NAME _____ PHONE _____

P.O. BOX _____ MAIL STOP _____

ADDRESS _____

CITY _____ STATE _____ ZIP CODE _____

COMPANY _____

ORGANIZATION _____

version 01.00.00

SYNTHESIS GUIDEBOOK SURVEY

NAME _____ PHONE _____

P. O. BOX _____ MAIL STOP _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

COMPANY _____

ORGANIZATION _____

SHADED FIELDS ARE OPTIONAL FOR SURVEY BUT MUST BE COMPLETED TO REGISTER FOR UPDATES.

FOLD HERE

Please give us your opinion after using the Synthesis Guidebook.

Also, if you are not registered to receive updates, please complete the form above.

Fold, tape, and mail.

The Consortium is working continually to improve Synthesis. After you have actually used it, please give us your opinion by completing the short survey (on reverse side). It should take only a couple of minutes. We'll send you the new Software Productivity Consortium Poster in return.

FOLD HERE

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 7702 HERNDON VA

POSTAGE WILL BE PAID BY ADDRESSEE

TECHNOLOGY TRANSFER CLEARINGHOUSE
SOFTWARE PRODUCTIVITY CONSORTIUM, INC.
SPC BUILDING
2214 ROCK HILL RD
HERNDON VA 22070-9858

SYNTHESIS GUIDEBOOK USAGE SURVEY

After using the Synthesis Guidebook, please tell us how well we did by completing this short survey form

Rate the Synthesis Guidebook for

	ADEQUATE				INADEQUATE		
	Totally	Very	Barely	Borderline	Somewhat	Mostly	Totally
Utility of approach	☐	☐	☐	☐	☐	☐	☐
Understandability	☐	☐	☐	☐	☐	☐	☐
Effective organization	☐	☐	☐	☐	☐	☐	☐
Helpfulness of examples	☐	☐	☐	☐	☐	☐	☐
Ease of tailoring	☐	☐	☐	☐	☐	☐	☐
Overall rating	☐	☐	☐	☐	☐	☐	☐

In what application domain(s) did you use Synthesis?

_____

_____

How does Synthesis compare to your organization's current *(or previous)* approach to software development?

Much Worse	Worse	Same	Better	Much Better	Can't Tell Yet
☐	☐	☐	☐	☐	☐

Please briefly describe that approach:

_____

_____

Will you continue to use or expand use of Synthesis?

Yes ☐ No ☐

Why: _____

_____

Other Comments: _____

_____

How was Domain Engineering effort funded?

IR&D	Bid & Proposal	Contract	Overhead
☐	☐	☐	☐

What kind(s) of applications did Synthesis support?

IR&D	Bid & Proposal	Contract	Overhead
☐	☐	☐	☐

Estimated size (in lines of code) of typical application:

Less than 10K	10K to 100K	100K to 500K	500K to 1M	Above 1M
☐	☐	☐	☐	☐

Estimated number of people who used the Guidebook.

1 to 3	4 to 6	7 to 10	11 to 20	More than 20
☐	☐	☐	☐	☐

How did you learn Synthesis?

☐ Self-taught *(guidebook)* ☐ Consortium training

What are the two most important improvements the Consrotium should make to the Synthesis Guidebook?

_____

_____

My primary responsibility is:

[ ] Project Management
[ ] Systems Requirements Analysis/Design
[ ] Software Requirements/Design/Implementation
☐ Support/V&V
☐ Other _____

SOFTWARE PRODUCTIVITY CONSORTIUM

version 01.(X).(X)

SUPPLEMENTARY

INFORMATION

Example 2. "Aircraft_Status_Display" Decision Group

Aircraft_Status_Display		
Decision	**Value Space**	**Description**
Situation	CWS	Indicated collision warning situation.
Partition	enum of (ID, UID)	Indicated aircraft partition. ID is "identified"; UID is "unidentified".
PT_Color	enum of (red, orange, green, yellow, white, blue, black, pink, purple, indigo, violet)	Icon color for the potential threat.
PT_Blink	boolean	Designates whether the potential threat icon should blink for this collision warning situation. True means blink; false means do not blink.
PT_Fill	boolean	Icon filling for the potential_threat involved in this collision warning situation. True means fill the icon (i.e., color the icon interior); false means do not fill in the icon.
...		

Each decision and each decision group must be given a unique identifier by which it can be referred to in the definition of adaptable work products. Each decision group has one list or table that is labeled with a mnemonic appropriate to the group. The group is a set of related decisions. Each entry is a separately-decidable decision that has its own distinct mnemonic label (the column labeled Decision in Example 1), a specification of allowed values that can resolve the decision (the Value Space column), and a short explanation of the meaning of the decision (the column labeled Description). Table DE.3.2.1-1 shows a possible set of alternative decision types that you can use to specify the value space for a decision. Other types may be added as needed.